NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND TEST OF THE CROSS-FORMAT SCHEMA PROTOCOL (XFSP) FOR NETWORKED VIRTUAL ENVIRONMENTS

by

Ekrem Serin

March 2003

Thesis Advisor:	Don Brutzman
Co Advisor:	Joseph Sullivan
Second Reader:	Curt Blais

This thesis done in cooperation with the MOVES Institute
Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Design and Test of The Cross-Format Schema For Networked Virtual Environments			**5. FUNDING NUMBERS**
6. AUTHOR Ekrem Serin			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT

A Networked Virtual Environment (Net-VE) is a distributed software system in which multiple users interact with each other in real time even though these users may be located around the world [Zyda 99]. Net-VEs gained first attention through a variety of DOD and Academic research projects. After release of the multiplayer game DOOM, the gaming industry captured the idea of interactive multiplayer games. Today there are many popular Internet-based multiplayer games available.

Effective networking of diverse entities and systems is a common problem for Networked Virtual Environments. In order to communicate with other entities a variety of communication protocols are used. Historically these communication protocols are "hard coded" into the software system and all nodes that participate in the environment must identically implement the protocols to interact with others. These communication protocols require authoring and compiling by a trained programmer. When the compiling process is introduced to the networked virtual environment, it detracts the extensibility and dynamicism of the system.

This thesis presents the design and development of a Networked Virtual Environment model that uses Cross Format Schema Protocol (XFSP). With this work we show that a networked simulation can work for 24 hours a day and 7 days a week with an extensible schema based networking protocol and it is not necessary to hard code and compile the protocols into the networked virtual environments. Furthermore, this thesis presents a general automatic protocol handler for schema-defined XML document or message. Additionally, this work concludes with idea that protocols can be loaded and extended at runtime, and can be created with different-fidelity resolutions, resulting in swapping at runtime based on distributed state.

14. SUBJECT TERMS Networked Virtual Environments, Cross Format Schema Protocol (XFSP), XML, XSD, SOAP, HLA, NPSNET-V, JXTA, XML Serialization.			**15. NUMBER OF PAGES** 149
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DESIGN AND TEST OF THE CROSS FORMAT SCHEMA PROTOCOL (XFSP) FOR NETWORKED VIRTUAL ENVIRONMENTS.

Ekrem Serin
Lieutenant Junior Grade, Turkish Navy
B. S., Turkish Naval Academy, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2003

Author: Ekrem Serin

Approved by:

Don Brutzman
Thesis Advisor

Joseph Sullivan
Co Advisor

Curt Blais
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A Networked Virtual Environment (Net-VE) is a distributed software system in which multiple users interact with each other in real time even though these users may be located around the world [Zyda 99]. Net-VEs gained first attention through a variety of DOD and Academic research projects. After release of the multiplayer game DOOM, the gaming industry captured the idea of interactive multiplayer games. Today there are many popular Internet-based multiplayer games available.

Effective networking of diverse entities and systems is a common problem for Networked Virtual Environments. In order to communicate with other entities a variety of communication protocols are used. Historically these communication protocols are "hard coded" into the software system and all nodes that participate in the environment must identically implement the protocols to interact with others. These communication protocols require authoring and compiling by a trained programmer. When the compiling process is introduced to the networked virtual environment, it detracts the extensibility and dynamicism of the system.

This thesis presents the design and development of a Networked Virtual Environment model that uses Cross Format Schema Protocol (XFSP). With this work we show that a networked simulation can work for 24 hours a day and 7 days a week with an extensible schema based networking protocol and it is not necessary to hard code and compile the protocols into the networked virtual environments. Furthermore, this thesis presents a general automatic protocol handler for schema-defined XML document or message. Additionally, this work concludes with idea that protocols can be loaded and extended at runtime, and can be created with different-fidelity resolutions, resulting in swapping at runtime based on distributed state.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Dr. Don Brutzman, CDR Joseph Sullivan and Curt Blais for their motivation and support throughout this study.

To Don McGregor and Andrzej Kapolka, I thank you for your guidance, wisdom, patience and enthusiasm throughout this project. Your efforts have given me an invaluable learning experience.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

The term Networked Virtual Environment (Net-VE) is defined as follows *"A networked virtual environment is a software system in which multiple users interact with each other in real-time, even though those users may be located around the world. These environments aim to provide users with a sense of realism by incorporating realistic 3D graphics and stereo sound, to create an immersive experience"* by Michael Zyda [Zyda 99]. A new networking framework for run-time extensible networked virtual environments is presented in this thesis.

Run-Time Extensible Virtual Environments differ from traditional Virtual Environments through the capabilities of run-time discovery and usage of new object types and behaviors. Traditional VEs can only operate with objects, behaviors and protocols that are present when the VE is started; if any kind of new object, behavior or protocol needs to be added to the architecture, the environment must be stopped, compiled and restarted. NPSNET-V is a Run-Time Extensible Virtual Environment implemented by the Naval Postgraduate School (NPS) that uses run-time extensibility for new object and behavior discovery.

Historically communication protocols that are used in Net-VEs are hard coded into the software system and all entities that participate in the environment need to implement the protocols to interact with others. Introducing a new application layer protocol requires off-line authoring and compiling by a trained programmer. This compiling process detracts from the extensibility and dynamicism of Net-VEs.

In RTEVE networking protocols can be loaded and extended at runtime. Furthermore, protocols can be created with different fidelity resolutions which can be swapped at runtime, based on the network state. Since the protocols can be tailored to best support the requirements of a particular environment, they can enhance network performance. These improvements can be made adaptively at run-time by a non-professional programmer or by software agents.

B. MOTIVATION

With the dramatic increase in computing and network speed over the past few years, Net-VEs have become more widespread. This enhancement introduced the non-professional programmers into the networked virtual environment area. The main motivation behind this thesis was to enable non-adept programmers to tailor their networking protocols by using the Extensible Markup Language (XML).
The NPSNET-V architecture is used to adapt the designed and implemented XML-based networking protocol.

NPSNET-V is used as a framework for development and research in dynamically extensible, large-scale virtual environments (LSVEs). The main property of this architecture is run-time discovery of new object types and their behaviors. A limitation of NPSNET-V architecture is networking protocols which cannot be tailored without recompiling and restarting. This limitation detracts the goal of run-time extensibility within this architecture.

C. OBJECTIVES

This thesis serves two purposes; design and implementation of a user-tailored networking protocol which is called Cross Format Schema Protocol (XFSP), and presenting that protocol with different schemas for use in a networked virtual environment.

The networking protocol (using XFSP) determines the state changes in entities which participate in a Net-VE and is used to exchange state information (e.g. position, orientation etc.) between entities.

In a Net-VE, entities issue packets when their states are changed or when they want to keep their states alive in all other participating users. To establish the communication between participants, all users (the ones that want to exchange data) must agree on the same protocol. The major differences between XFSP and hard-coded networking protocols is run-time extensibility and flexible ease of use.

As it is discussed before, XFSP is based on defining packet formats with XML-Schema language. The process behind XFSP can be called XML Serialization, or XML marshalling. The basic idea is similar to creating a Document Object Model (DOM) pipe between participants. When a user receives a packet into which an XML tree is serialized, he or she can build a DOM object tree back from the packet and can retrieve the needed data. The position of the data in that DOM object tree is defined by XML-Schema.

Determining the semantics between the DOM tree and entity state is considered to have a computational complexity of NP-Hard and is not examined in this thesis. The semantic process is defined as being able to know which information to retrieve from an information source in order to reflect the output in a Net-VE. Run-time extensibility of a semantic process is hard to accomplish and requires thoughtful design. Facilitating the expression and distribution of protocol syntax is a major step forward nevertheless which enables rapid exploration of efficient and effective protocol semantics.

Ordinarily, XML is not a compact way to express the data. Messages written in XML are much larger than a binary equivalent. The technique that is used to overcome this problem is replacing tags with binary tokens. When an XML tree is parsed to serialize into an output stream, the tags that mark up the data are replaced with their binary equivalents. The end result is a more compact serialized XML tree.

As it is discussed before, the basic idea behind XFSP was XML-Serialization. With this approach, XFSP can be used in any application which needs transactions via XML documents such as XML-RPC (XML Remote Procedure Call), XKMS (XML Key Management Services), XML-DSig (XML Digital Signatures) and XML-Enc (XML Encryption). XFSP can present those transactions in a more compact way.

In this thesis the usage of XFSP in a Networked Virtual Environment to exchange state information between entities by using the idea of creating DOM pipes between participants is presented. Also with this thesis, a PDU Server (Protocol Datagram Unit Server) and a PDU Capturer are implemented for servers. The PDU Server program is used to continuously send state information from a previously recorded simulation and

the PDU Capturer program is used to capture the packets issued by entities from a simulation.

In order to test this new protocol, experiments are conducted across a wide-area network (WAN) with George Mason University (GMU) and on a local-area network (LAN).

The results of these experiments are discussed in following chapters.

D. THESIS ORGANIZATION

Seven chapters comprise this research:

- *Chapter I–Introduction*: Identifies the purpose and motivation behind conducting this research. Establishes the goals for the thesis.
- *Chapter II–Related Work and Background*: Provides information on Networked Virtual Environments, NPSNET-V, XML, XSD, JXTA, SOAP, HLA, DOM4J, DREN and Abilene/Internet2 projects.
- *Chapter III–Design and Implementation of XFSP*: Describes the general system structure, software components and implementation process of XFSP.
- *Chapter IV–PDU Farm Implementation*: Describes the general system structure, software components and implementation process of PDU Server and PDU Capturer programs.
- *Chapter V–Binary X3D*: Describes the general system structure, software components and implementation process of Binary X3D program.
- *Chapter VI–Data Collection and Analysis*: Explains the results and collected metrics from the conducted experiments.
- *Chapter VII–Conclusion and Recommendations*: Explains the conclusions and provides recommendations regarding possible future work.

II. RELATED WORK AND BACKGROUND

A. INTRODUCTION

This chapter provides an overview to Net-VEs, NPSNET-V, XML, XML-Schema, Simple Object Access Protocol (SOAP), JXTA, High Level Architecture (HLA), DOM4J, Defense Research and Engineering Network (DREN) Internet2/Abilene/Next Generation Internet (NGI).

B. NETWORKED VIRTUAL ENVIRONMENTS (NET-VES)

A Networked Virtual Environment (Net-VE) is a distributed software component with a computer generated simulated space in which multiple users interact with each other in real time. The main motivation behind Net-VEs is run-time collaboration between participants.

Each participant uses his or her computer in order to access and collaborate with the displayed 3D content. In virtual environments, users are represented by one or more entities called avatars. These avatars give the users the illusion of being located in that simulated space where the immersive experience is presented. Net-VEs are used for multiple purposes varying from education and entertainment to training. From a military point of view, Net-VEs are a cost effective way to train the war fighters and mimic the war scenarios.

By simulating scenarios in a networked environment, we have opportunity to examine the interactions between participants. This examination helps decision makers to present new tactics and doctrines for future scenarios.

A Net-VE system consists of four basic components [Zyda 99].

- Graphic Engines and Displays
- Communication and Control Devices
- Processing Systems
- Data Network

In this chapter the bottlenecks created by these components in Net-VE systems are examined.

1. Graphic Engines and Displays

Graphic engines, also called graphics cards, are used to process the 3D content generated by some Application Programming Interface (API) (e.g. OpenGL, DirectX) in order to make it displayable through some output device (such as a computer monitor). Display devices vary from 14" Cathode Ray Tube (CRT) to fully immersive head-mounted displays. These devices open the three-dimensional (3D) window of the virtual environment to the participants. Traditional displays offer only limited immersion to the user where they can be distracted by outside light and peripheral vision. For higher immersion, users often use Head Mounted Displays (HMDs). HMDs present images directly in front of a user's eyes and block out almost all external light. Another example for immersive displays is CAVE. CAVE is a five sided cube where the user stands at the center and the images are projected to the sides that give a highly immersive experience to the participant.

With the dramatic increase in silicon technology, today we are able to access very fast and powerful graphic cards which can render millions of triangles per second with pixel shading capability. If the bottlenecks of generating Net-VE content are examined from a network programmer point of view, limitations are not the graphic engines and displays.

2. Communication and Control Devices

The second attribute of a Net-VE system is Communication and Control Devices, which are used to manipulate the virtual world objects and communicate with other participants in the environment. For manipulation purposes, users typically depend on keyboard and mouse. Mouse is a key device for navigation in the virtual environment to control the speed of travel and perform other interactions. Incidentally, a keyboard is also utilized for typing textual communication and can be used for complicated or less-common operations in the environment.

Although the keyboard and mouse are the most common control devices, they are not the only ones. For game applications the common devices are joysticks, wheel-drives or derivatives of these devices. For applications that need more precise manipulation, datagloves can be a good candidate. In the near future, with the progress in speech synthesizers, voice-recognition applications will be valuable candidates for control purposes.

To achieve full immersion from the Networked Virtual Environment, users should be able to communicate with each other. Historically this is done by using keyboard and textual communication which is inconvenient for the participants. For example, with today's technology Voice over IP (VoIP) can be an effective way for meeting participant communication and collaboration objectives in an integrated fashion.

3. Processing Systems

The third component of a Networked Virtual Environment is processing systems. A Net-VE system needs a considerable amount of processing capacity [Brutzman 97]. The processor receives events from the user's input devices and computes how these inputs can change hosted entity's positions within the virtual environment as well as the location of other entities within the environment [Zyda 99]. The processor also decides how and when to notify other users about these state changes.

In a Net-VE system, Central Processing Unit (CPU) can be easily overwhelmed by different computation needs. A physically based modeling exemplar of an F-16 aircraft can present this heavy-weight computation by processing air dynamics and kinematics when precisely modeling the entity. These flight coefficients and quaternion mathematic equations [Cooke 92] may be the bottleneck in designing the system.

As discussed before, a Net-VE system is a distributed software architecture in which multiple users interact with each other. In a large-scale Net-VE with many participants scenario, without a partitioned network, processing system will be first bottleneck in creating this content. In this large-scale Net-VE example, the bottleneck can be examined in two ways. First is the buffer (memory) limitation of the computer system

in which the environment is simulated, and the second is the cycle limitation of the processing unit. This discussion bases the use of XFSP in a Net-VE system where sufficient computational horsepower exists to allow users to customize their application layer protocol. Fortunately, this overhead is shown to be low, allowing commodity PC and laptop devices to use XFSP.

4. Data Network

The last component of a Net-VE system is Data Network. This component will be examined in more depth to base the discussion on implementing Cross Format Schema Protocol (XFSP).

For several years one of the major factors that limited research into large-scale distributed virtual worlds was immature network technology [Macedonia 97]. Immature network technology constrained Net-VEs to operate on local-area networks (LANs), which limited the number of participating hosts in geographical scope.

In order to expand this geographical scope and increase number of participating hosts, large-scale Net-VEs should operate over Internet , wide-area networks (WAN), and exploit its resources. Furthermore, operation over WAN will bring some discussion on communication aspects such as bandwidth, latency, distribution schemes, and reliability.

a. Bandwidth

Bandwidth is defined as the width of the usable spectrum that is available for data signal transmission. The spectrum refers to the range of frequencies that a signal contains [Stallings 00]. The bandwidth of the communication link depends on the material (media) on which it operates. It can be twisted pair, coaxial cable, fiber optic or air. The highest bandwidth among those is in fiber optic cable.

Bandwidth plays an essential role in determining the richness and size of a Net-VE. As the number of participating hosts in a Net-VE system increases so does the bandwidth requirement. Additionally, by the nature of virtual environment, a Net-VE

system will demand a considerable amount of bandwidth to support video, audio and the exchange of 3D content in real time.

With today's technology, the networks that have gigabits per second bandwidth over Ethernet links or Asynchronous Transfer Mode (ATM) cells are already available. Although this enormous bandwidth is present for some users, we cannot generalize all users. This efficient use of bandwidth must be considered.

The discussion on bandwidth determines the scope of target users. In a multiplayer game scenario, the target profile is often home users. Although there are many promising technologies such as Digital Subscriber Line (DSL) or Cable which can provide up to 2 Mbps download and 256 Kbps upload bandwidth, most of the home users are still limited to 56Kbps voice modems to join these multiplayer games. Average available bandwidth continues to increase each year.

Because of these reasons, bandwidth plays a crucial role in scalability of Net-VE systems. Application-layer protocols thus need to be customized properly to satisfy the participant's needs.

Well-defined techniques to handle data transmission over the network links is a prerequisite for supporting internetworked computer graphics [Brutzman 97].

b. Latency

Network latency is the amount of time required to transfer application data from one point to another [Zyda 99]. Latency controls the Net-VE's interactive and dynamic nature, directly impacting the realism of the environment by determining how up-to-date is the information received.

In order to give the players the illusion of being immersed in a networked virtual environment, the limits of human perception must be rigorously considered. For a distributed environment that emulates the real world, Net-VE architecture must deliver the packets with sufficiently minimal latency, and generate 3D images at 30-60 Hz to guarantee the illusion of reality [Macedonia 97].

Latency is a problematic network component, and Net-VE designers can usually do very little about it, because latency is a combination of many constraining factors. The first factor to consider is the speed of light. Electromagnetic or optical transmission cannot travel faster than that speed. Actually the real propagation speed depends on the medium in which the data signal travels, and generally speaking for electromagnetic or optical media, it is 2/3 of the speed of light in a vacuum. The second factor is the delay introduced by the network. In today's packet-switched network topology data passes through multiple repeater, router and switch hops during its journey from source to destination. Most significant are routers which route the packets over the network. Each router introduces queuing and processing delays for each packet that increases the total amount of delay. The third delay to consider occurs between the operating system and network interface hardware on application computers. It takes some amount of time for the data to travel from Operating System (OS) kernel to network hardware even before it reaches the network.

c. Jitter

Given these many factors, users may think that the latency is static and cannot change in time. This is not correct statement. Latency is considered to be a stochastic process and cannot be determined precisely, furthermore it will change from packet to packet. This change is called "jitter" and is used to define the variation of delays. Because of jitter, state changes in a Net-VE system cannot be received at a steady rate, thereby degenerating the smooth perception by introducing jerky behavior. In order to cope with latency effects, a variety of dead reckoning algorithms are proposed by researchers. Specifically, dead reckoning depends on predictive modeling which estimates current state of the entity based on previous state, elapsed time and other factors.

d. Distribution Schemes

Distribution determines the way to transfer data packets from source host to destination hosts. There are three ways to transmit data over the network to the users.

These are unicast, broadcast and multicast. Distribution scheme is correlated with scalability and reliability. Where scalability is defined by the richness of the virtual environment and reliability is defined by the data loss rate.

(1) Unicast: Unicast is a point-to-point communication between two end-users. In this scheme only the recipient host and intermediate routers need to spend computational cycles during the journey of the packet. With this nature, unicast distribution does not scale well for networked virtual environments. In an environment with N participants, each host must open N-1 connections and send the same data multiple times onto the network. The connection and bandwidth complexity can be considered as O (n^2) which is clearly an expensive proposition when sending high bandwidth streams such as audio and video to multiple users.

(2) Broadcast: Broadcasting is at the opposite end of the spectrum from unicast distribution scheme. Broadcast messages reach every host on a local-area network and demand a response from each operating system. This is an extremely inefficient way to distribute packets. Even a few small global broadcasts could bring the Internet to its knees. Imagine what could happen if a real-time video feed were copied to six million Internet users, whether they wanted to watch it or not [Harold 00]. That is the reason that broadcasting is prohibited from passing across the switches and routers. As it is seen, broadcasting does not scale well for large-scale networked virtual environments and limited to the local area networks, furthermore it does not reach Internet at all.

(3) Multicast: There is a middle ground between point-to-point communication and broadcasting to the whole world. One way to do this is to create static connection trees. This was the solution used by some conferencing systems (e.g. CU-See Me). In this example data is fed from the originating site to other servers, which replicate it to the other servers, which eventually replicate it to the clients [Harold 00]. In this architecture the connection tree does not reflect the best possible network topology to distribute the packets and the hooks into the tree must be done manually. It would be better to allow the routers to determine the best path for transmitting one-to-many or many-to-many type transmission of data. This is where the multicasting comes in.

Multicasting is based on the idea of groups. Each host can subscribe to any multicast group in order to send and receive data to or from that group. Multicasting is done at the hardware level by informing the network interface card (NIC) to monitor any specific group. This feature is extremely important since a single high bandwidth stream can still reach an arbitrary number of hosts, but computational load is only seen on hosts which explicitly subscribe to the multicast channel [Brutzman 97].

In order to have multicasting capability and creating one-to-many or many-to-many distribution scheme, the routers on the way should be multicast enabled. With today's Internet architecture there are still problems. Most of the routers are not multicast enabled; specifically they are not "mrouters". Current architecture of multicasting can be described as the following model. This model describes how end systems are to send and receive the multicast packets.

- IP-style semantics : A source can send multicast packets at any time, with no need to register or to schedule transmission. IP multicast is based on User Datagram Protocol (UDP), so the packets are delivered using a best-effort delivery
- Open groups : Sources only need to know a multicast address. They don't need to know group membership, and they do not need to be a member of the multicast group to which they are sending. A group can have any number of sources.
- Dynamic Groups : Multicast group members can join or leave a multicast group at will. There is no need to register, synchronize or negotiate with a centralized group management entity.

This standard IP multicast model is an end-system specification and does not discuss requirements for how the network should perform the multicast routing. Also it does not propose any mechanisms for providing quality of service (QoS), security or address allocation. Actually the major problem here is the routing itself.

There were many efforts to establish multicast connectivity or routing. One of them used handcrafted tunnels across the Internet where the multicast

stream is encapsulated with unicast packets. The idea behind this approach was establishing UDP channel, IP-encapsulated tunnel, between sub-networks and encapsulating multicast data with UDP packets. This model was called MBone. The MBone carried its first worldwide event when 20 sites received audio from the meeting of the IETF in San Diego in 1992 [MBoneInternet2]. The most significant achievement was the deployment of a virtual multicast network. The multicast routing function was provided by workstations running a daemon process called *mrouted*, which received unicast-encapsulated multicast packets on an incoming interface and then forwarding the packets over the appropriate set of outgoing interfaces. Routing decisions were made using Distance Vector Multicast Routing Protocol (DVMRP), which is based on *reverse shortest path tree*s.

Actually these discussions brought the idea of intra and inter domain multicasting. For intradomain multicasting new solutions showed themselves, such as MOSPF (Multicast Extensions to Open Shortest Path First) where MOSPF routers flooded the group membership information to other MOSPF routers or newer protocols PIM-DM (Protocol Independent Multicast Dense Mode) and PIM-SM (Protocol Independent Multicast Sparse Mode) where they use an existing unicast routing table to build a multicasting table. The difference between dense and sparse mode is the mechanism that they use for multicasting, such as *broadcast-and-prune* or *explicit join*. Broadcast-and-prune method is considered to be the dense, and explicit join is the sparse mode.

For interdomain multicasting the solution was BGP (Border Gateway Protocol) which supports the routing by reliably exchanging network reachability information between border gateways where a network administrator can run any protocol within his/her domain.

Despite all these accomplishments, the multicasting is still not widely deployed; most of the routers drop every multicasting packet that hits its incoming interface. Multicasting is not the only problem with today's Internet architecture, QoS issues are still not solved, IPv6 is still not supported and there are problems with

bandwidth issues. These topics will be discussed in Internet2 / NGI (Next Generation Internet) section.

e. Reliability

Reliability typically measures how much data is lost by the network during its journey from source to destination [Zyda 99]. In reliable systems it is assumed that when data is sent it is always received. Reliable transport protocols use acknowledgement and error-recovery schemes. Unfortunately this introduces considerable amount of delay and inefficient use of bandwidth to the system. As we saw before, real-time systems need fast transmission and high-bandwidth capacity. With these acknowledgement, error recovery and congestion control components it is not practical to implement both reliable and real-time networked virtual environments. Consequently, most of the Net-VE systems use unreliable transport protocol (UDP over IP based networks). These systems have been designed to recover from a lost packet and it is not crucial to lose a state information packet when the next one is going to be received a short time later.

The Transmission Control Protocol (TCP) is the reliable transport protocol that sits on top of the Internet Protocol (IP). TCP uses acknowledgement, sequencing, error-recovery and flow-control schemes to achieve its functionality. Another transport protocol is User Datagram Protocol (UDP), which is very different than TCP. It offers best effort delivery without any promise of sequencing.

Although it depends on the architecture that the user wants to implement, in most of large scale Net-VEs hybrid systems are used. In hybrid systems, both of the transport protocols run at the same time for different purposes. TCP can be used for managerial and UDP can be used for state exchange objectives.

Multicasting is implemented by using UDP, however there are many efforts in developing reliable (partial) and scalable multicast services by using designated receivers and other ideas [Pullen 95] and [Pullen 00].

C. OVERVIEW OF A RTEVE (NPSNET-V)

The NPSNET program of Naval Postgraduate School started in 1990 as a research platform for networked virtual environment technology. It is now its fifth iteration known as NPSNET-V [Salles 02]. NPSNET-V is implemented to meet the following goals.

- Run-time extensibility of both content and applications
- Scalability in world complexity and number of participants
- Composability of heterogenous content and applications

As stated by McGregor and Kapolka, implementers of NPSNET-V, the dream of NPSNET-V is for it to be "… a framework for fully distributed, component-based, persistent, networked virtual worlds, extensible at runtime and scalable to infinite size on the Internet" [McGregor 01],[Salles 02].

A full description of NPSNET-V architecture is too extensive for the scope of this work; therefore only an overview of the components that play an essential role in XFSP scope will be covered. Readers can refer to [McGregor 01], [Kapolka 02], [Salles 02] and [Capps 01] for more detailed description.

1. Components

NPSNET-V is a component-based framework and used to build virtual worlds by combining modules at run-time. The run-time extensibility of NPSNET-V does not depend on prior knowledge of individual modules. Furthermore, new behaviors (new components) can be added to the system "on-the-fly".

NPSNET-V can divided into five functional areas [Salles 02]

- Configuration Files: the blueprints of NPSNET-V
- Communications: the communication infrastructure of NPSNET-V
- Database: the database of all necessary data for NPSNET-V
- Components: the functional code modules used to build VE
- Temporal: time coordination system

Network communication architecture is examined next with regard to the new framework called XFSP.

2. Network Communication Architecture

NPSNET-V is implemented to handle unicast, broadcast and multicast channels as well as using reliable or unreliable transport protocols.

There are three types of communications performed in NPSNET-V architecture.

- Administrative Communications which deal with required exchanges of necessary components. TCP type connections are used.
- Object Communication; which deals with passing of object code, modules and terrain data from HTTP or specialized servers. They can be transmitted over reliable or unreliable connection schemes.
- Entity Communication which deals with state exchanges. Both reliable and unreliable transport schemes might be used. For the scalability of the environment unreliable transport protocol over multicast distribution scheme is preferred. Until recently Distributed Interactive Simulation (DIS) (IEEE 1278.1) was used to exchange state information between entities. With this work, a new type of entity state exchange protocol, XFSP, is introduced to the NPSNET-V architecture. XFSP exploits the idea of XML-Serialization and DOM-Pipe generation. Design and implementation of XFSP will be examined in Chapter III.

D. XML

Extensible Markup Language (XML) is a markup language used to describe the structure of data in meaningful ways. The most well known XML applications are web-related, but there are many other non-web based applications where XML is considered to be very useful for solving specific problems. An example for this type of usage is the financial transactions between different businesses.

XML can be used in many applications varying from databases to extensible 3D environments. Actually it can be said that XML is to be used in any application where "data" is processed, which results in "every" application.

With today's evolving technology, XML is considered to be the solution for many problems in computer world. In my opinion the biggest problem is the interoperability between non-homogenous systems. The differences between these non-homogenous systems can be summarized in four major points.

- Computer Architecture
- Operating Systems
- Data Structures
- Programming Language

Being a W3C standard, XML is a simple text document which describes the actual data with meta-data. Any programming language running on any platform can parse this text document and interpret it to its internal data structure. With this nature, XML is the solution for system-integration and interoperability problems in non-homogenous systems. Reader can refer to [Hunter 01] and [W3CXML] for more information on this topic.

E. XSD

An XML Schema is the modeling document which defines the structure of an XML document. Schemas are used to validate the XML documents.

XML Schema uses the same syntax that XML uses, it fully supports the Namespace Recommendation and allows creation of complex and reusable content models with the idea of object inheritance and type substitution.

The fundamental idea behind validation is to create XML documents that they can be shared by multiple users without any conflict when they follow the same rules that the schema defines. Any well-formed XML document can be validated against any schema.

There are many open-source XML Schema parsers and XML validators which can be downloaded from web-sites. The basic process of XML validation is shown in Figure 2.1.

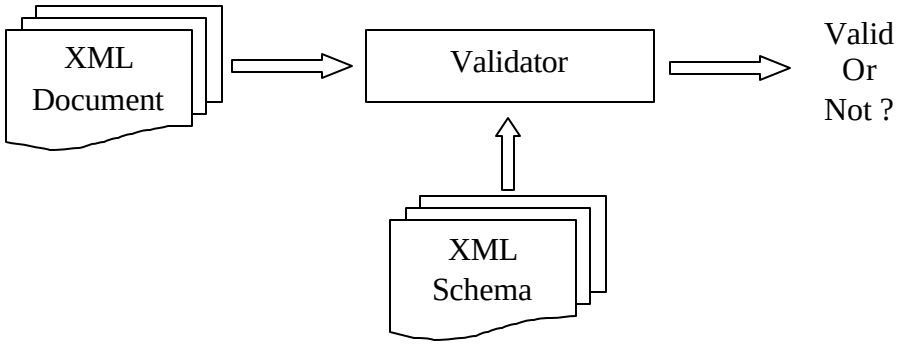

Figure 2.1: XML Validation Using Schema

In XFSP, XML Schema is used to define the application layer protocol format in a Net-VE platform for participating users. The complete process of XFSP will be discussed in Chapter III. For now, the fundamental idea is creating a schema parser over top of a non-commercial schema parser where it can parse the schema to properly define the protocol syntax.

For more information about XML Schema and schema parsers refer to [W3CSchema].

F. JXTA

JXTA Project is an open-source technology which is about communication, collaborations and sharing. The JXTA applications vary from instant messaging to transactional web services to interactive gaming. This technology provides an open, generalized platform for building peer-to-peer (P2P) applications.

JXTA provides a set of simple and flexible mechanisms for enabling devices such as cell phones, wireless PDAs, PCs and servers to act as peers on a virtual network. As a peer, any connected device can establish ad hoc networks to find, access and use the vast resources of other peers on the network regardless of location [JXTA].

JXTA protocols establish a virtual network on top of existing networks, hiding their underlying complexity. Unlike traditional client-server networks which rely on a centralized point of message transport, a JXTA virtual network allows peers to interact with other peers or resource directly, even across firewalls or different network boundaries. In order to cross firewall problems, JXTA uses HTTP tunneling over port 80. Most of firewall administrators consider port 80 as safe and allow communication connections on that port. JXTA exploits this idea and crosses firewall boundaries.

Today, companies in diverse industries such as wireless telecommunications, government, entertainment, financial services and educations have started using and evaluating the JXTA technology.

For example in the telecommunications industry, JXTA technology enables the creation of ad hoc wireless networks. In the media and entertainment industry, developers are exploring the use of P2P technologies for a new generation of interactive and participative games.

The power of JXTA comes in building virtual networks quickly for short-term projects as well as for long ones without needing to create separate, costly and complex infrastructures.

JXTA may not seem a solution for all of the networking infrastructure needs, but it is a promising technology for creating ad hoc virtual networks for distributed computing.

G. SOAP

In order to understand SOAP (Simple Object Access Protocol) we must have a firm understanding of basic concept of *web services.* A web service is a network accessible interface to application functionality, built using existing Internet technologies [SOAP]. Web services is an interface positioned between the application code and the user of that code. It acts as an abstraction layer, separating the platform and programming language-specific details of how the application code is actually invoked. This process of abstraction is shown in Figure 2.2.

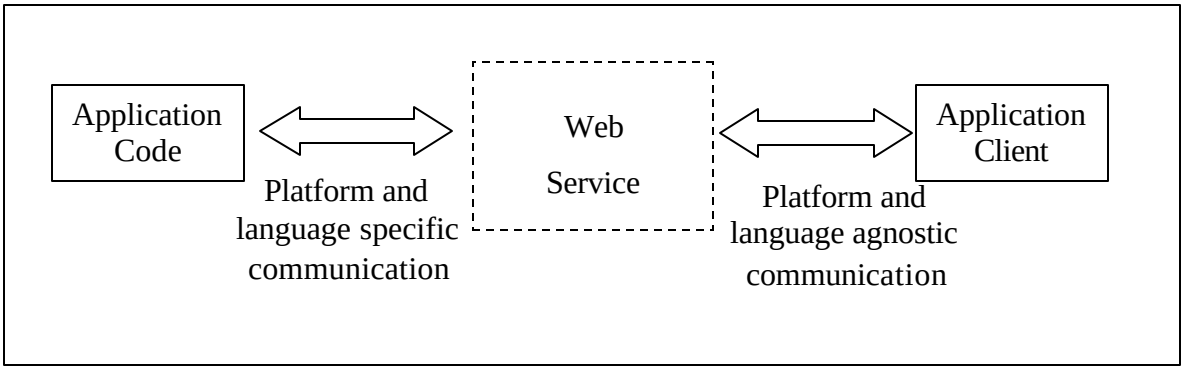

Figure 2.2: SOAP Web-Services Abstraction [SOAP]

SOAP's place in the web services is as a standardized packaging protocol for the messages shared by different applications. The SOAP specification defines nothing more than a simple XML-based *envelope* for the information being transferred and a set of rules for translating application and platform-specific data types into XML representations (XML Marshalling and XML Unmarshalling).

SOAP is just an XML document relying on XML standards like XML Schema and XML Namespaces for its definition and functionality. SOAP is a basic *XML Messaging* where applications exchange information. It provides a flexible way for applications to communicate.

An XML message can be anything: a purchase order, a request for current stock price or current position of friendly forces. The XML messaging process is shown in Figure 2.3.

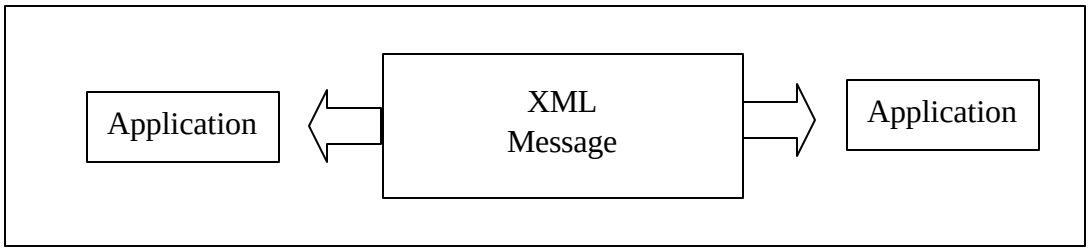

Figure 2.3: XML Messaging [SOAP]

Because XML is not tied to any particular application, operating system or programming language, XML messages can be used in all environments. A Unix-based C

program can create an XML document representing a message and send it to a Windows Java program where it can be interpreted without any conflict.

The fundamental idea is that two applications, regardless of operating system, programming language or any type of technical implementation detail, may share the information using nothing more then a simple message encoded in a way that both applications understand.

This was at the heart of XFSP, where SOAP-like XML messaging is used to transmit entity state information. In order to gain from the bandwidth, XML element and attribute names are replaced by binary short tags and XML document is serialized into packets. The complete process of XFSP will be discussed in Chapter III.

H. HLA

The High Level Architecture (HLA) is a software architecture that can be considered as the glue which allows users to combine computer simulations into a larger simulation [HLA 00]. HLA helps to create a big simulation from pieces; the fundamental idea is component-based simulation.

For instance, there may be a need to combine simulations in several different regions with simulators running on different machines, such as Fast Patrol Boats, Frigate Divisions, Amphibious Ships, Tactical Air Support Maritime Operations (TASMO) Aircrafts to create a navy battle simulation. HLA combines these standalone simulations into a single and combined simulation with the ability to extend it in the future by adding new simulations.

In order to be familiar with the HLA framework, some new terms are introduced below.

Federation: Federation is the combined simulation system created from the existing simulations.

Federate: Federate is the each simulation that is combined to form a federation.

Federation Execution: Federation Execution is a session of a federation executing together.

A federation contains a supporting software called the Run-time Infrastructure (RTI), a common object model for the data to be exchanged between federates (simulations) in a federation called the Federation Object Model (FOM) and a number of federates [HLA 00].

In the HLA framework, one federate might represent one platform such as a Fast Patrol Boat, or a federate might represent an entire Fast Patrol Boat Division. From the perspective of HLA, a federate is defined by its single point of attachment to the RTI.

As shown in Figure 2.4, a federate might model some number of entities, or it might have a different purpose such as just being a data collector by passively receiving data and generating none.

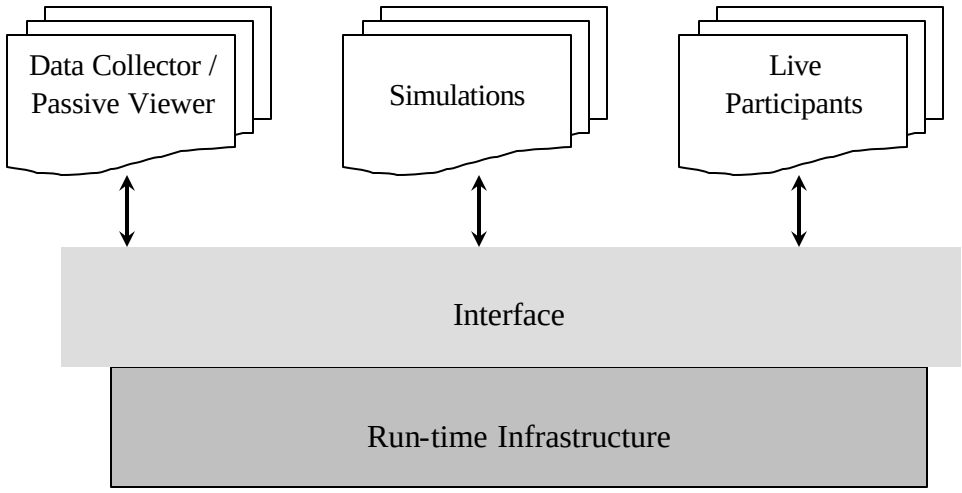

Figure 2.4: Software Components in HLA [HLA 00]

The fundamental idea behind the HLA is software reuse. Design goals of the HLA are listed below.

- It should be possible to decompose a large simulation problem into smaller parts. Smaller parts are easier to define, build correctly and verify.

- It should be possible to combine the resulting smaller simulations into a larger simulation system
- The functions that are generic to component-based simulations systems should be separated from specific simulations. The resulting generic infrastructure should be reusable from the one simulation system to the next.
- The interface between simulations and generic infrastructure should isolate the simulations from the changes in the technologies [HLA].

With regard to these goals, the HLA is foremost a software architecture, rather than a particular implementation of its infrastructure. It contains a variety of different implementations. Consequently, it is defined not by the software, but by a set of documents.

Currently the HLA is an IEEE standard as the IEEE 1516 specification. For more information about the HLA the reader can refer to [HLA 00], [HLADMSO] and [HLAIEEE].

I. DOM4J

DOM4J is a Java toolkit for writing XML processing applications, with its own tree-based model for XML documents, inspired by the XPath data model [DOM4JCook 01]. It has both event-based and tree-based modes, supports evaluation of XPath expressions against the document tree, and also has an implementation of its own tree model that supports the DOM.

The current implementation of XFSP uses DOM4J API in order to accomplish XML processing. XML processing includes a number of sub-components such as XML Serialization, XML Deserialization, XML Schema Parsing and XPath Addressing. DOM4J is chosen to be the API for this project for the following reasons.

- Open-Source: It is an open-source project where the source of the complete API can be downloaded, inspected and customized (as necessary) according to the user needs.
- Easy-to-Use: It is a Java-based API, easy-to-use and intuitive for a Java programmer. It takes best features from DOM and SAX and puts them together.
- Standards Compliant: It fully supports DOM and SAX together with existing Java platform standards such as Java 2 Collections and J2EE.
- Complete XPath Integration: Complete XPath support is integrated into the API. XPath is the ideal technology for navigating around XML documents simply and easily.
- Handle Very Large Documents: It is fast and efficient with small memory overhead for parsing, where it can process large XML documents with the support of XPath, XSLT and XML Query.

With DOM4J, users are able to create their own XML tree implementations by simply providing a DocumentFactory implementation where it can support dynamic data binding to XML tree nodes.

For further details about DOM4J API, reader should refer to DOM4J Cookbook [DOM4JCook 01] and on-line documentation [DOM4JOnline].

J. DREN

The Defense Research and Engineering Network (DREN) is DoD's high-performance network. The DREN is a robust, high-speed network that provides connectivity among geographically dispersed user sites and shared resource centers. The networking services of DREN are provided by a contract service where the service provider has built DREN as virtual private network (VPN) over a public infrastructure. The DREN provides digital data transfer services between defined service delivery points (SDPs). SDPs are specified in terms of wide area networking (WAN) bandwidth access,

network protocols (Multi Protocol Label Switching, IP, ATM) and local connection interfaces [DREN 03].

DREN currently supports bandwidths from DS-3 (45 Mbps) at user sites to OC-12 (622 Mbps) at selected centers. Future bandwidths will scale to OC-192 (10 Gbps) and beyond. The sites to be connected by DREN services may be at virtually any point in the continental Unites States, Alaska and Hawaii.

Incorporating the best operational capabilities of both the DoD and the commercial telecommunications infrastructure, DREN is the official DoD long-haul network for computational research and testing. DREN enables many scientists and engineers at defense laboratories, test centers, universities and industrial sites to use high-performance computing resources.

K. INTERNET2/ABILENE/NGI

The rapid growth of Internet in the number of hosts, number of users, traffic level, traffic nature and topology complexity has often resulted in the degradation of Quality of Service (QoS) available to the end-users. Some replication techniques, such as caching of recently accessed information and deliberate mirroring are used to reduce the Internet traffic. Nevertheless, problems are common.

The growing interest in multimedia applications had huge impact both on telecommunication networks and the Internet. With current Internet's *best-effort delivery* it seemed not quite possible to support applications which need differentiated services, real-time streaming or real-time collaboration. These developments led the education and research community to think about a Next Generation Internet (NGI). The Internet2 initiative provided a forum for the universities and research communities interested in NGI activity [Ghonaimy 99].

The Internet2 project aims at serving the education and research community and is mainly a joint project among universities mostly in United States. A release project is the Next Generation Internet (NGI) which is envisaged to cope with demanding new applications in all fields. The ultimate objectives of Internet2 are:

- Developing a fast all-optical network
- Using more efficient switches and routers where IPv6 is supported
- Developing the new multicasting technology
- Putting an emphasis on end-to-end QoS

The first stage of Internet2 relied on the vBNS (the very high-speed Backbone Network Service), but later developed its own backbone called Abilene, announced in April 1998 [Ghonaimy 99].

The Abilene backbone network provides high-performance, best-effort delivery for U.S. nationwide connectivity to Internet2 universities and their institutions. The Naval Postgraduate School is one of the members of this community. Abilene is a pure *packet-over-SONET* (POS) network providing coast-to-coast OC-48 (2.4 Gbps) IP transit. Connectors attach to one of the POPs (Point of Presence) with either POS or IP-over-ATM access circuits, running at OC-3 (155Mbps), OC-12 (622 Mbps) or OC-48 (2.4 Gbps) speed. Current logical connectivity is shown in Figure 2.5.

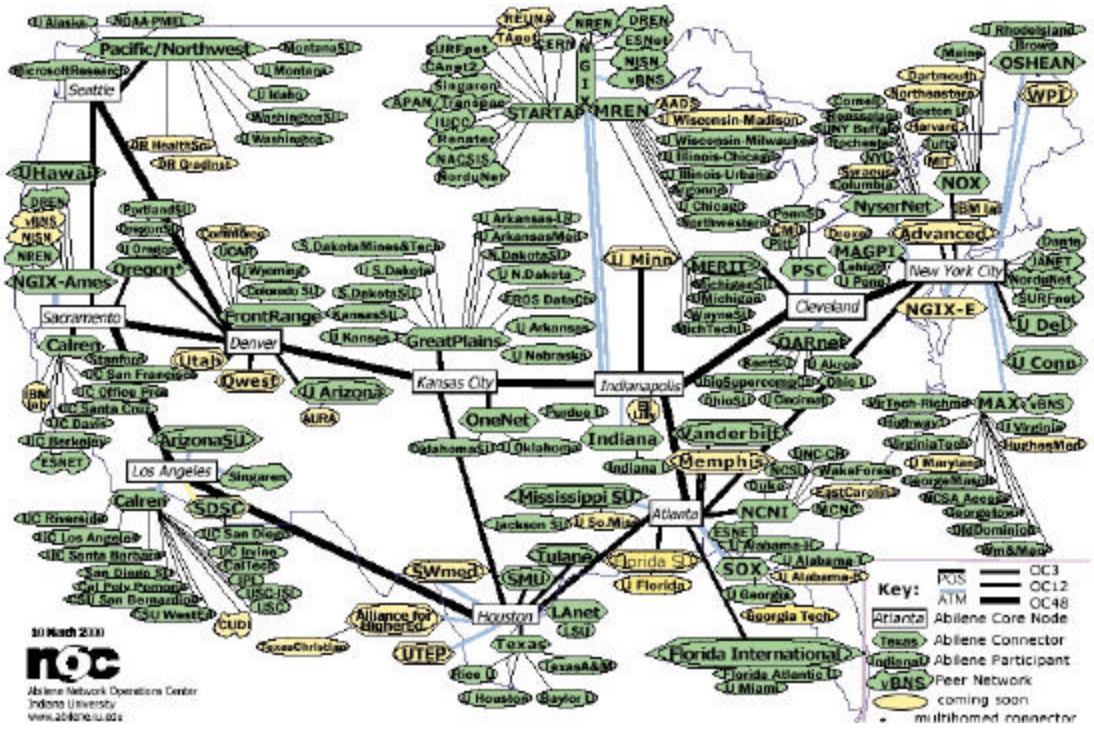

Figure 2.5: Abilene Network Logical Map [Abilene 00]

With Abilene, Internet2 and NGI projects, the research community is trying to find answers to the needs of real-time interactive applications which typically support human-to-human collaboration or human-to-machine remote control. This demand of interactivity imposes stringent worst-case delay, jitter and loss requirements on the underlying network service. These applications often demand worst-case network performance bounds that must be maintained on any time interval longer than a few milliseconds.

In order to accomplish these achievements, researchers are trying to extend current Internet technology with the ones that discussed above. Current Internet technology is a single-service network where all packets are treated almost same. Many research efforts are contributed to bring solution to the QoS problems and provide a better technology. With all these efforts it seems quite promising to have a high-speed Internet which fully supports IPv6 and differentiated services in the near term.

L. RELATED WORK

1. An Automated Approach to Distributed Interactive Simulation (DIS) Entity Development

Michael Canterbury's thesis tried to solve the limitation of DIS protocol for large-scale Net-VEs. In that thesis, Canterbury targeted the DIS support for real-time, simulated engagements of more than 1000 entities. In order to solve that problem, Canterbury refined the existing DIS protocol and optimized the form and content of DIS network traffic [Canterbury 95].

The approach taken was to design and build a protocol development tool which was accomplished in three phases. In the first phase, a modified Backus-Naur Form (BNF) grammar was formulated for use in modeling DIS data elements. In the second phase, the grammar was applied to the PDUs and data types specified in DIS standard. In the last phase, a tool, the DIS *Protocol Support Utility*, was developed as a means to automate the process of authoring, editing and implementing refinements to the DIS protocol.

As a result of this effort, Canterbury specified the data elements depicted in DIS standard by using a BNF-like grammar. With the implementation of *Protocol Support Utility*, the generated grammar was parsed and program source code associated with each data element automatically generated. The conclusion that the framework reached was [Canterbury 95],

- A modified BNF grammar can be used to describe DIS protocol entities.
- A simple grammar might be used to model the data elements associated with a complex protocol.
- Though the modified BNF syntax was suitable for modeling the PDUs and data entities associated with DIS, it was not well suited for modeling the methods or implementation details needed in processing those entities.

For further information about run-time protocol extensibility, reader should refer to [Zeswitz 93], [Canterbury 95] and [Fischer 01].

2. Wireless Access Protocol (WAP) Wireless Markup Language (WML) Specification

Wireless Application Protocol (WAP) is a result of continuous work to define an industry-wide specification for developing applications that operate over wireless communication networks. The WAP Forum, originally founded by Ericsson, Motorola, Nokia, and Unwired PlanetWML was formed to create the global wireless protocol specification that works across differing wireless network technology types, for adoption by appropriate industry standards bodies. To enable operators and manufacturers to meet the challenges in advanced services, differentiation and fast/flexible service creation, WAP defines a set of protocols in transport, session and application layers [WAP 99].

The WML specification designed by WAP Forum defines a compact binary representation of the XML. The binary format was designed to allow for compact transmission with no loss of functionality or semantic information. The format is designed to preserve the element structure of XML, allowing a browser to skip unknown elements or attributes. The binary format encodes the parsed physical form of an XML

document, i.e., the structure and content of the document entities. Meta-information, including the document type definition and conditional sections, is removed when the document is converted to the binary format.

Based on XML, Wireless Markup Language (WML) is intended for use in specifying content and user interface for narrowband devices, including cellular phones and pagers. WML includes four major functional areas [WAPW3C]:

- Text presentation and layout: WML, including image support and a variety of formatting and layout commands.
- Deck/card organizational metaphor: All information in WML is organized into a collection of cards and decks.
- Inter-card navigation and linking: WML includes support for explicitly managing the navigation between cards and decks.
- String parameterization and state management: All WML decks can be parameterized, using a state model.

3. Compressing XML with Multiplexed Hierarchical Prediction by Partial Match (PPM) Models (XMLPPM)

XMLPPM is a data compression program that compresses XML files. It is a combination of the Prediction by Partial Match (PPM) algorithm for text compression, and an approach to modeling tree-structured data called Multiplexed Hierarchical Modeling (MHM) developed by Cheney [XMLPPM].

The main idea behind XMLPPM is leveraging the work that a Simple API for XML (SAX) parser does by encoding the sequence of events. An implemented decoder decodes these events, and reconstitutes an XML document equivalent to the original. Alternatively, the decoder acts as a SAX parser, parsing encoded event sequences instead of text, and sending those events directly to the application.

In XMLPPM, a single byte event encoding is used to encode element start tags, end tags and attribute names and to indicate events such as “begin/end characters, begin/end comment”, and so on. The encoder and decoder maintain consistent symbol

tables; whenever a new symbol is encountered, the encoder sends the symbol name and the decoder enters it to the table [XMLPPM].

Additional to the SAX type encoder/decoder, in XMLPPM a *Multiplexed Hierarchical Modeling* is used. This technique employed two basic ideas: multiplexing several text compression models based on XML's syntactic structure (one model for element structure, on for attributes and so on) and injecting hierarchical element structure symbols into the multiplexed models. With these techniques XMLPPM claims that it can compress XML files from 5 to 30% better than any existing text or XML-specific compressors. Further information about XMLPPM can be found at [XMLPPM] and [XMLPPM 03].

4. XMill

XMill is an XML-conscious compressor that compresses XML documents by using the redundancy and standard text compression approaches. XMill combined with `gzip` compresses XML data about 10% better then `gzip` on equivalent non-XML forms; further improvement (up to 50%) is possible with user assistance in the form of complex command-line parameters [XMill 00].

XMill shows that XML-conscious compression can do better than text compression alone. XMill applies three principals to compress XML data [XMill 00].

- Separate structure from data: The structure consisting of XML tags and attributes is compressed separately from the data.
- Group related data items: Data items are grouped into *containers*, and each container compressed separately.
- Applying semantic compressors: XMill can apply specialized semantic compressors to different containers.

However, XMill's base transformation has several drawbacks, it requires user assistance to achieve best compression, and it wins only if the data set is large, typically

over 20 KBytes because of the additional bookkeeping overhead and the fact that small data containers are poorly compressed by `gzip`.

For further information about generic and XML-specific compression schemes reader should refer to [ZLib], [BinXML], [XMill 03] and [Millau].

M. SUMMARY

In this chapter Net-VEs, NPSNET-V, XML, XML-Schema, SOAP, JXTA, HLA, DOM4J, DREN, Internet2/Abilene/NGI are discussed. Additionally, related works such as An Automated Approach to Distributed Interactive Simulation (DIS) Entity Development, WML, XMLPPM and XMill are introduced to provide the background and reasons for this thesis research.

THIS PAGE INTENTIONALLY LEFT BLANK

III. CROSS FORMAT SCHEMA PROTOCOL (XFSP)

A. INTRODUCTION

This chapter introduces the description, design and implementation detail of Cross Format Schema Protocol (XFSP). This general XML compression scheme can be used for automatic production of binary network protocols, and binary file formats, for any data structures specified in an XML Schema.

B. OVERVIEW OF XFSP

The idea of creating flexible and run-time extensible application layer protocols inspired the implementation of XFSP. The question to be answered was *"How can a run-time extensible application layer protocol be implemented?"* After preliminary searches in published papers and different software architectures, XML appeared to be an excellent approach for describing data structures. Specifically, XML-Schema can be used to define the protocol syntax. Because the protocols are considered to be the definition language describing the agreement between end-users and end-systems, they can be described well by XML-Schema using its internal as well as user defined data structures.

In order to accomplish this, a schema parser is implemented on top of an open-source XML parser called DOM4J [DOM4JOnline 03]. That schema parser parses the provided schema and creates entries in a table for each element and attribute defined in the schema document. The need for this process is to be able to send the XML documents in a more compact way by exploiting the protocol (i.e.*"agreement"*) notion. Initial tests showed that a short XML document can go beyond the MTU (Maximum Transmission Unit; i.e. 1500 bytes for Ethernet) very easily. The solution for that problem came with binary XML or XML Serialization. Instead of sending the XML document as a serialized text, a replacement algorithm might be used that replaces the element and attribute names with short tag numbers to serialize the data.

Because the protocols are defined as an agreement between user applications and if users agree on using the same protocol via the same schema, then there is no need to send the whole name for each element and attribute. This may raise the question why a

DIS-like (Distributed Interactive Simulation) [DIS] protocol is not used instead of XFSP. Actually the answer for this question is obvious. First, in DIS-like protocols the syntax of the protocol is static and all of the data must be in their previously specified places, so there is no way to send a subset or selected section of the whole protocol. The second reason is that such protocols are not run-time extensible. The syntax of the protocol cannot be changed while the application is running. The last reason is the serialized data can be converted easily back to the text and can be used in web-service applications like SOAP.

With XFSP two main ideas are implemented: changing the protocol syntax at run-time and XML Serialization / Deserialization. The problem domain is targeted to meet the needs of non-homogenously distributed users, where non-homogeneity is primarily described as the bandwidth distribution. Although fast networks are already present in today's networking infrastructures, users are still non-homogenously distributed and cannot always access high bandwidth. The idea of changing the protocol at run-time to meet the needs of the different users at different fidelity levels is considered as a valuable problem to address.

The following sections cover the design and implementation details of XFSP.

C. PROTOCOL DESCRIPTION VIA XML SCHEMA

The key elements of a network protocol can be described as:

- Syntax: Describes the format and the position of data.
- Semantics: Describes the functional connection between application and data.

For the XFSP project, semantics is not targeted to be solved and is generally considered to be NP-Hard, because the semantic definition needs a knowledge domain and AI generation. As described before, the run-time extensible syntax is pointed out as the research question and targeted to be solved. To solve this problem, XML Schema is used to define the application-layer protocol between users and serialized XML data is sent as the payload.

An exemplar application layer protocol is shown in Figure 3.1, where it is defined by using XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with XML Spy v4.3 by Ekrem Serin (NPS) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="protocol">
        <xs:complexType>
            <xs:all>
                <xs:element name="header" type="HeaderType"/>
                <xs:element name="location" type="Vector3Double"/>
                <xs:element name="velocity" type="Vector3Float"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="HeaderType">
        <xs:all>
            <xs:element name="version" type="xs:short"/>
            <xs:element name="exerciseID" type="xs:byte"/>
            <xs:element name="pdutype" type="xs:int"/>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="Vector3Double">
        <xs:attribute name="x" type="xs:double"/>
        <xs:attribute name="y" type="xs:double"/>
        <xs:attribute name="z" type="xs:double"/>
    </xs:complexType>
    <xs:complexType name="Vector3Float">
        <xs:attribute name="x" type="xs:float"/>
        <xs:attribute name="y" type="xs:float"/>
        <xs:attribute name="z" type="xs:float"/>
    </xs:complexType>
</xs:schema>
```

Figure 3.1: A Simple Schema Document

This schema shows that there are three top level elements, *header*, *location* and *velocity*, that are connected to the root element called *protocol*. The data type of header is a custom type called *HeaderType* and the data types of location and velocity elements are custom type as well, *Vector3Double* and *Vector3Float* respectively. If the HeaderType is examined, it has three elements, *version* as short type, *exerciseID* as byte type and

pduType as *integer*. *Vector3Double* and *Vector3Float* define three attributes such as *x*, *y* and *z* as *double* or *float* types respectively.

If the protocol hierarchy tree is examined (Figure 3.2), it can be represented by a tree structure as follows. Elements are shown by boxes and attributes are represented by ellipses.

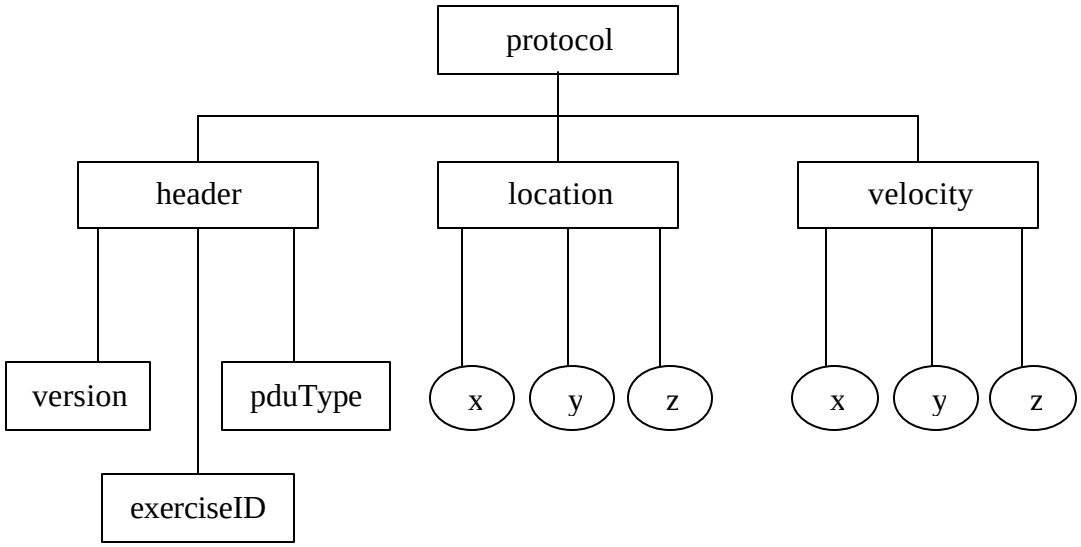

Figure 3.2: Tree Representation of Example Schema

The extensibility also shows itself in this tree representation, that the protocol can be described in different ways; i.e. the attribute and element locations do not have to be at the exact place in a PDU as it is in DIS. Another legitimate representation for the same protocol payload is shown in Figure 3.3.

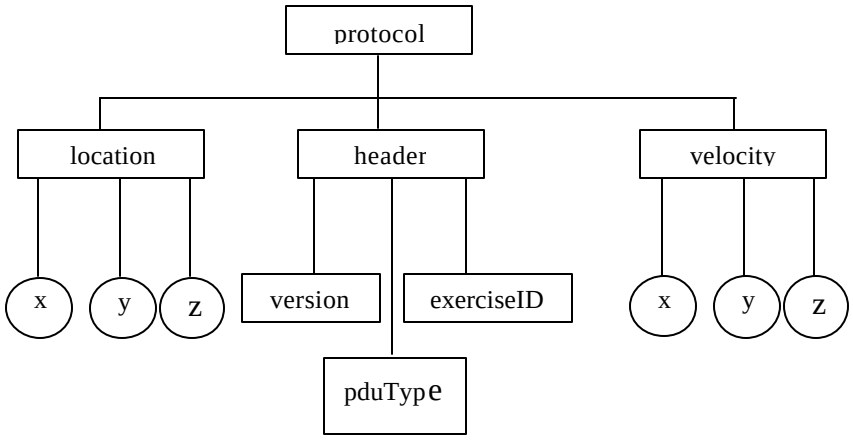

Figure 3.3: Alternate Tree Representation of Example Schema

Another area for extensibility might be the capability to send a subset of this protocol tree to users who do not need to know the whole data set. XFSP can also handle this selective transmission process.

The implementation detail of XFSP is examined in the following sections. The UML diagrams shown in Figure 3.4 and Figure 3.5 provide background information about the class hierarchy of the XFSP Project.

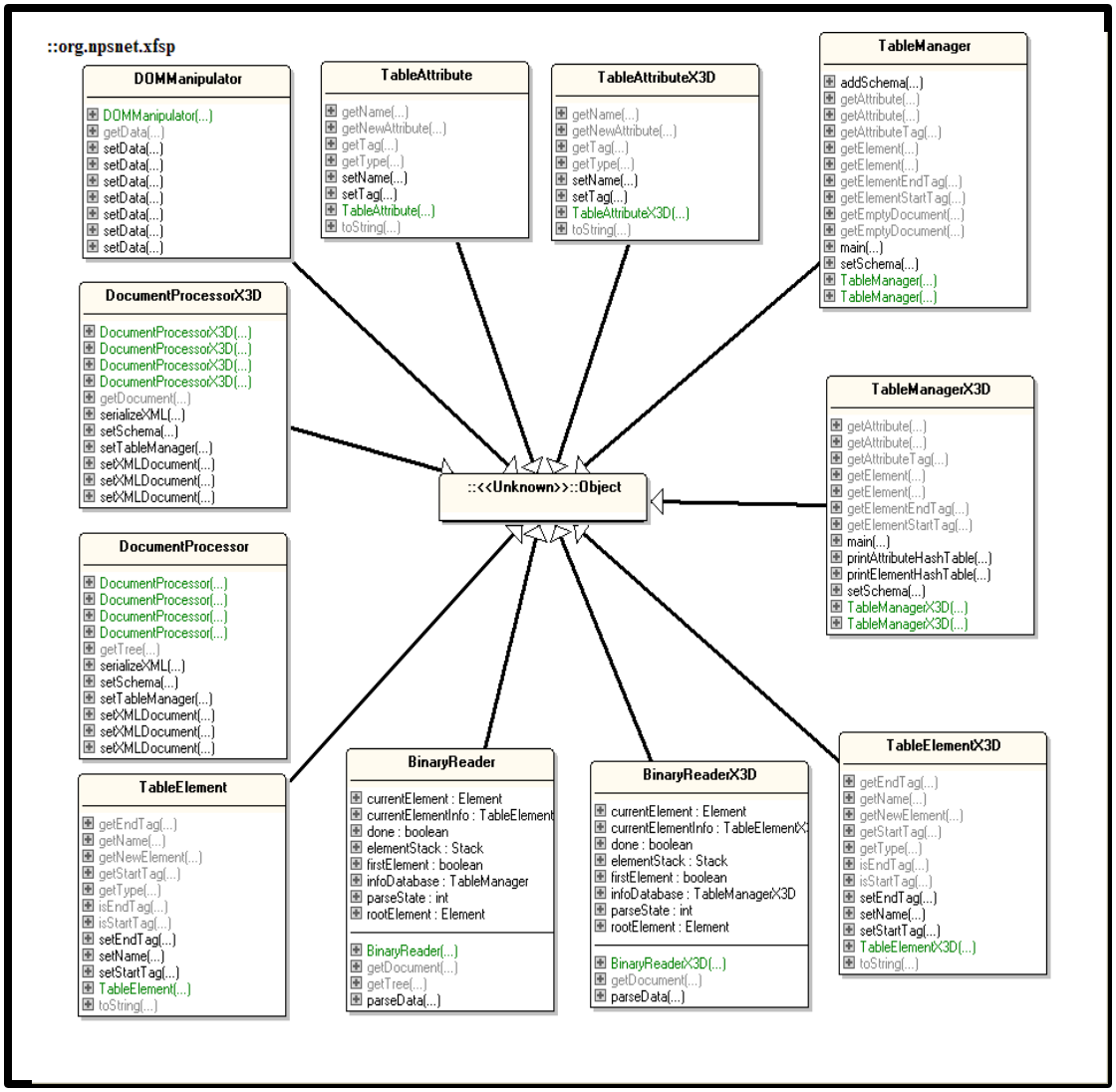

Figure 3.4: UML Diagram for Root Directory of XFSP (Generated by [ESS])

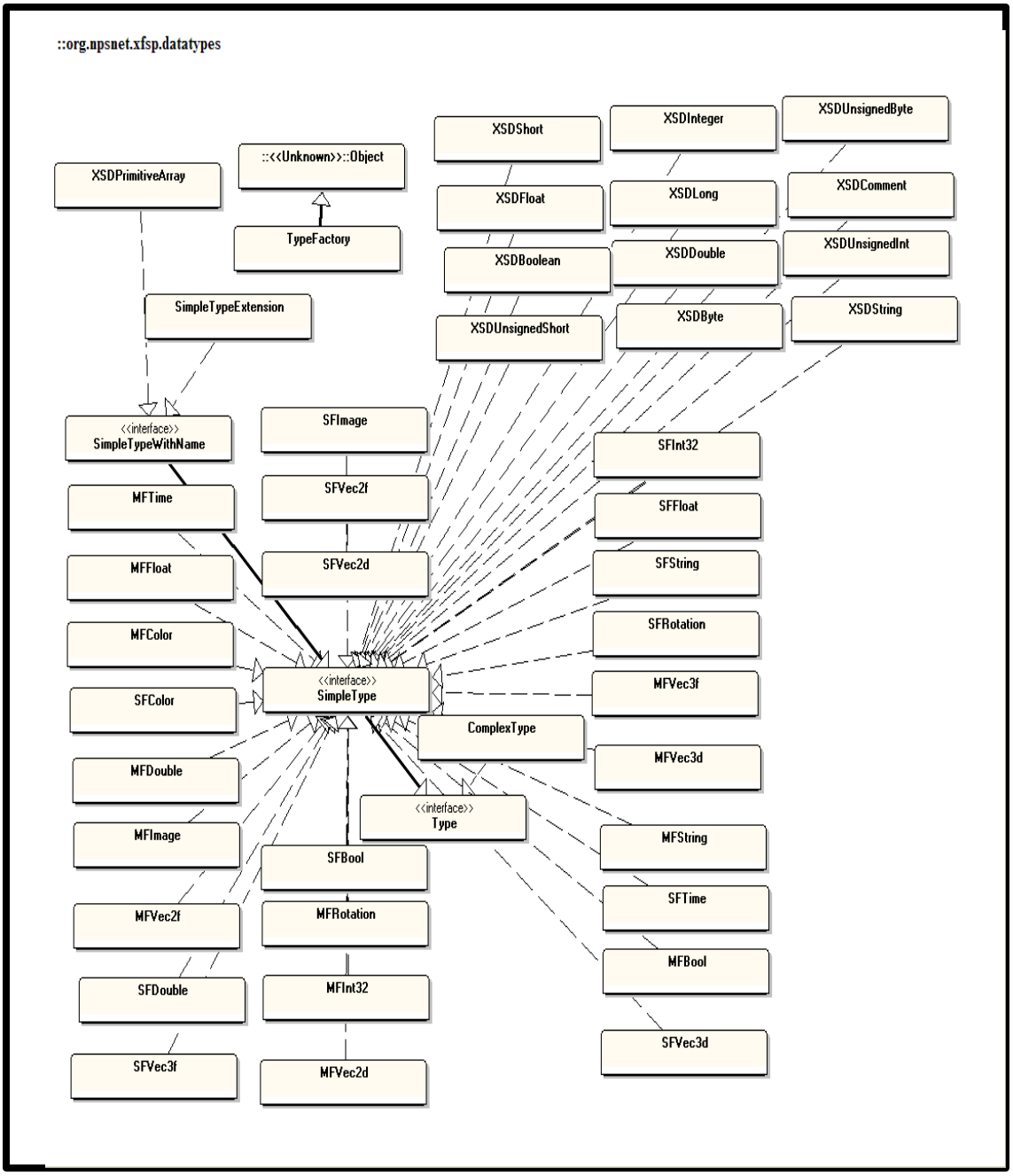

Figure 3.5: UML Diagram for Implemented Data Types corresponding to XML Schema and X3D (Generated by [ESS])

D. SCHEMA PARSING

As described before, the data-defining XML Schema is parsed to create the application layer protocol. The end product of the parsing process is a look up table where element and attribute names and associated data types can be retrieved.

The parsing process assigns a unique short number (i.e. token) for start and end tags of each element, as well as an element-specific token for each attribute. The process also determines the data types used by elements or attributes. The element or attribute names do not need to be unique; the schema parser is able to distinguish non-unique names.

If the previous schema document is examined for the parsing process, the resultant tables for element and attribute look ups would be as follows (Table 3.1 and Table 3.2).

Name	Start Tag Number	End Tag Number	Datatype
/protocol	10	11	Null
/protocol/header	12	13	ComplexType
/protocol/location	14	15	ComplexType
/protocol/velocity	16	17	ComplexType
/protocol/header/version	18	19	XSDShort
/protocol/header/exerciseID	20	21	XSDByte
/protocol/header/pduType	22	23	XSDInteger

Table 3.1: Element Look up Table Example

Name	Tag Number	Datatype
/protocol/location/@x	24	XSDDouble
/protocol/location/@y	25	XSDDouble
/protocol/location/@z	26	XSDDouble
/protocol/velocity/@x	27	XSDFloat
/protocol/velocity/@y	28	XSDFloat
/protocol/velocity/@z	29	XSDFloat

Table 3.2: Attribute Look up Table Example

The XPath addressing and tag numbers are used as keys for creating hash tables. These keys provide quick access to the elements and attributes during the serialization and deserialization processes. The table generation is done in *TableManager* class. This class also provides adding a new schema over the existing ones. In this way, new schema documents can be parsed at run-time when new protocols need to be introduced to the system.

Another feature of this class is providing an empty XML document (tree) according to the given schema. The capability to provide a sub-tree when the root of the requested sub-tree is passed as parameter is further provided.

The implemented parser cannot handle every schema document. Schema parsing operation is a difficult process and many cases need to be handled uniquely. One of the difficulties is assigning a unique name to every element and attribute. This is vital because it affects the correctness of the serialization and deserialization processes. A counter example where unique name assignment might be mishandled is shown in Figure 3.6.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="SameElementName">
      <xsd:complexType>
         <xsd:attribute name="SameAttrName" type="xsd:boolean"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="AnotherElement">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="SameElementName">
               <xsd:complexType>
                  <xsd:attribute name="SameAttrName" type="xsd:int"/>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Figure 3.6: Example Schema Demonstrating Valid Distinction of Dissimilar Elements with Identical Names

In this example there are two attributes with different data types but with the same name connected to different elements that have the same element name. Figure 3.6 demonstrates valid distinction of dissimilar elements with identical names. Discrimination of over-loaded element names occurs through context-sensitive inclusion of parent element inheritance during tokenization.

```
<xs:element name="protocol">
        <xs:complexType>
            <xs:all>
                <xs:element name="location_1" type="locationType_1"/>
                <xs:element name="location_2" type="locationType_2"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="locationType_1">
        <xs:sequence>
            <xs:element name="location" type="Vector3Float"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="locationType_2">
        <xs:sequence>
            <xs:element name="location" type="Vector3Double"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Vector3Float">
        <xs:attribute name="x" type="xs:float"/>
        <xs:attribute name="y" type="xs:float"/>
        <xs:attribute name="z" type="xs:float"/>
    </xs:complexType>
    <xs:complexType name="Vector3Double">
        <xs:attribute name="x" type="xs:double"/>
        <xs:attribute name="y" type="xs:float"/>
        <xs:attribute name="z" type="xs:float"/>
    </xs:complexType>
</xs:schema>
```

Figure 3.7: Automatically Amended, Intermediate Example Schema Demonstrating Valid Distinction

In Figure 3.7 the same conflict occurs in a different way where this conflict must be resolved before the tag assignment process. If the XPath representation of conflicted attributes is examined they can be represented as /protocol/location_1/location/@x (includes y and z) and /protocol/location_2/location/@x. Although they are bound to the

same element name, they are nonetheless unique and thus need to be differentiated. Attribute names are also fully qualified by parent elements and types to allow distinguishing them despite possible overriding of the same name by multiple many different attributes. The schema parser in XFSP handles many of these cases.

E. XML SERIALIZATION

The purpose of XML Serialization process is to send XML documents in a more compact way. In order to accomplish this, the attribute and element name replacement idea is used. The serialization can be considered as the preorder traversal of a given XML tree putting all node names and associated data to the output stream.

XFSP can serialize text-based XML documents as well as Document Object Model (DOM) trees. This feature comes with the open-source DOM4J API. *DocumentProcessor* class is implemented to serialize the XML documents or Document objects. The UML diagram of *DocumentProcessor* class is shown in Figure 3.4.

There are two main constructors for this class. First if the XML document to be serialized provides the schema that it implements, that information is used to build the look up table. Otherwise the *TableManager* must be constructed and passed to the *DocumentProcessor* before the serialization method is fired. The following XML document (Figure 3.8) implements the schema document provided before. This XML document is used as an example in describing the serialization process.

```
<?xml version="1.0" encoding="UTF-8"?>
<protocol xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="example.xsd">
    <location x="3.45" y="56.72" z="-10.1"/>
    <header>
        <exerciseID>1</exerciseID>
        <version>1</version>
        <pdutype>2</pdutype>
    </header>
    <velocity x="1.0" y="0.0" z="-0.7"/>
</protocol>
```

Figure 3.8: XML Document for a Sample Protocol

If element and attribute tag names are replaced in the serialization process with the short numbers looked up from the parsed schema, the above XML document can be represented as in Figure 3.9.

```
<?xml version="1.0" encoding="UTF-8"?>
<10 >
    <14 24="3.45" 25="56.72" 26="-10.1" 15>
    <12>
        <20>1<21>
        <18>1<19>
        <22>2<23>
    <13>
    <16 27="1.0" 28="0.0" 29="-0.7" 17>
<11>
```

Figure 3.9: XML Document with Replaced Tags

In order to signal specific conditions such as elements that have data or start and end of attribute reading, special tokens 0, 1 and 2 are used and these numbers are not assigned to any element or attribute in the schema parsing process. The serialization and deserialization processes are shown in Figures 3.10 and 3.11. The meanings of tokens and tags are also shown in those figures. Tokens are the short numbers signaling deserializer to change its state. Tags represent the name of the elements or attributes obtained from the previously generated look up table.

Tokenization and de-tokenization are handled in Serializer and Deserializer programs. The input for the Serializer is a raw XML document as shown in Figure 3.8. The output of the Deserializer is the same XML Document. The tokens and tags in the Figures 3.9, 3.10 and 3.11 are shown for the clarity and understandability purposes. The XML document is not tokenized before entering the Serializer. The tokens and tags are present during the transit of the document from the Serializer to the Deserializer.

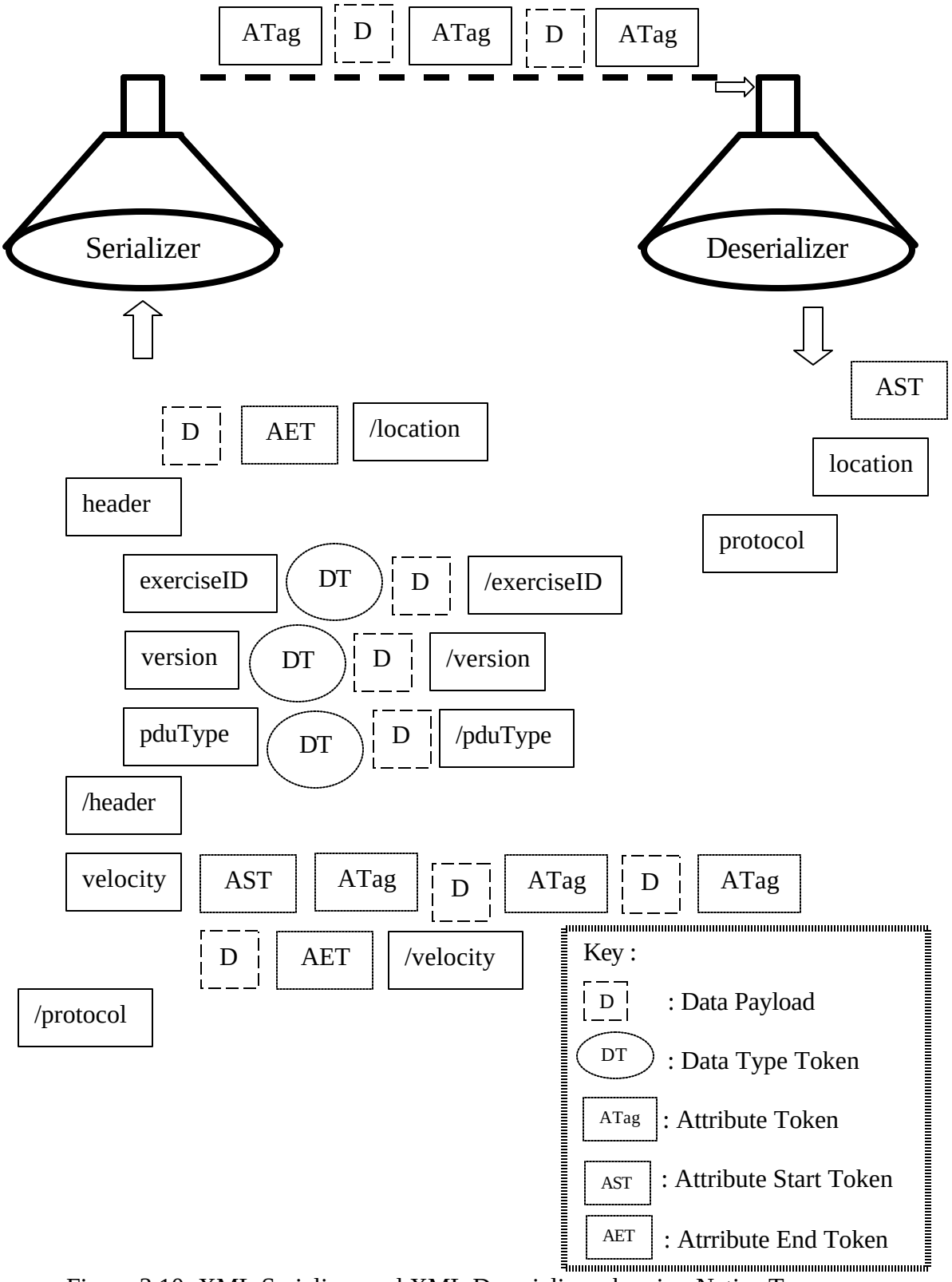

Figure 3.10: XML Serializer and XML Deserializer showing Native Tags

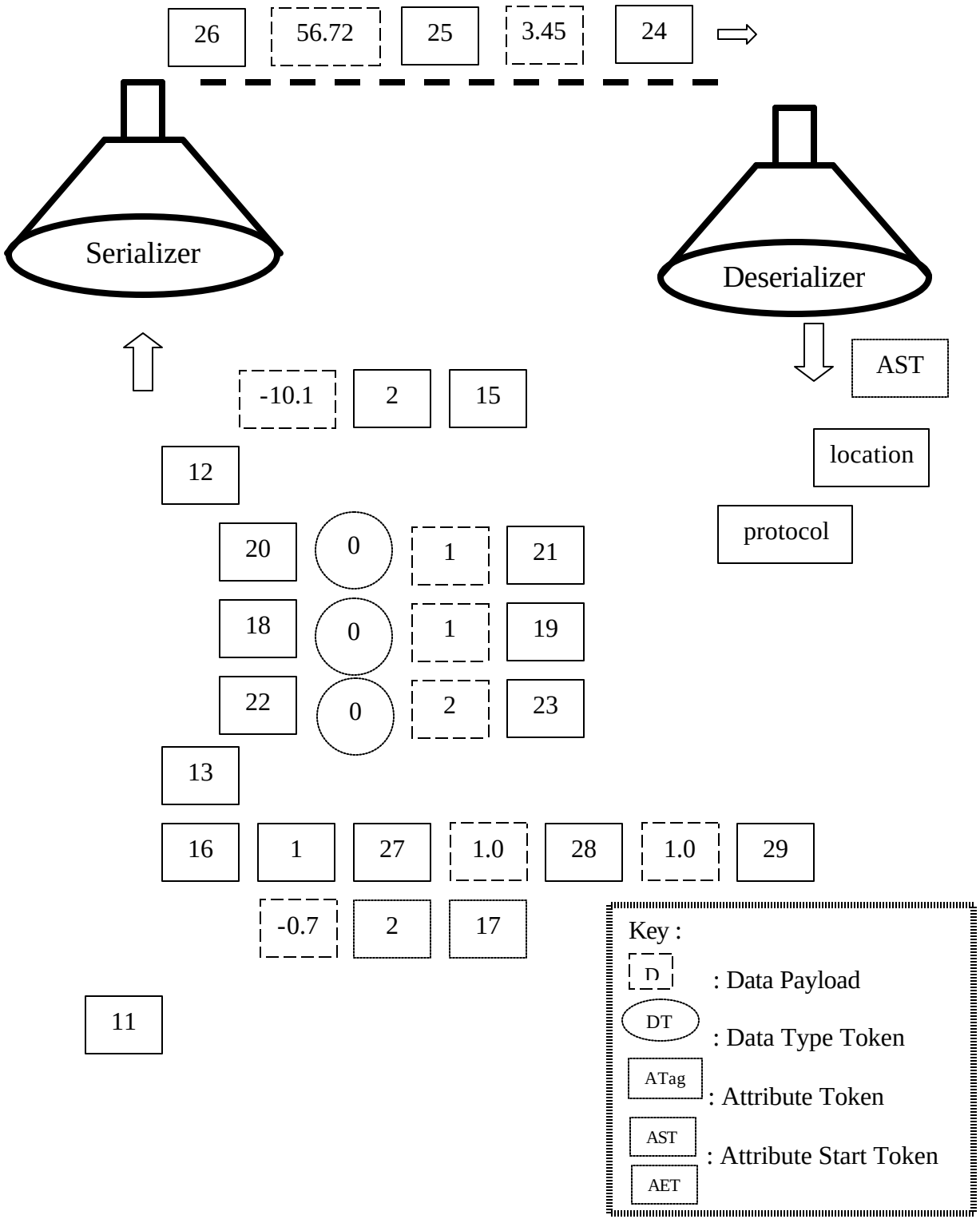

Figure 3.11: XML Serializer and XML Deserializer with Tokens and Tags Replaced by Short Numbers

In Figure 3.11, the Deserializer received the tokens 10, 14 and 1 and started building the XML tree gradually. The complete end product of XML Serializer is shown in Figure 3.12. The serialized document is interpreted from left to right and top to bottom.

10	14	1	24	3.45	25	56.72	26	-10.1	2	15

12	20	0	1	21	18	0	1	19	22	0

2	23	13	16	1	27	1.0	28	1.0	29	-0.7

2	17	11

Figure 3.12 : Serialized XML Document

In order to properly describe, validate and verify the designed algorithm, Communicating Finite State Machines (CFSMs) are used [CFSM]. Finite State Machines represent the actions taken by Serializer and Deserializer algorithms in serializing XML Documents or deserializing them back from the encoded stream. The finite state machine diagram of serialization process is shown in Figure 3.15. In this figure, states are represented by circles and the arrows between states represent the state changes. The valid transitions and failure of transitions are shown in Figure 3.13 and Figure 3.14

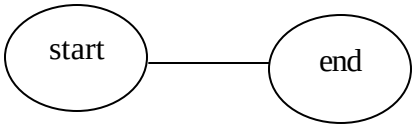

Figure 3.13: Valid Transition

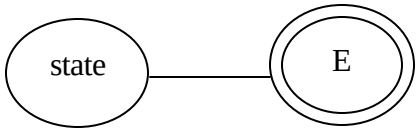

Figure 3.14: Transition Failure Resulting in Error

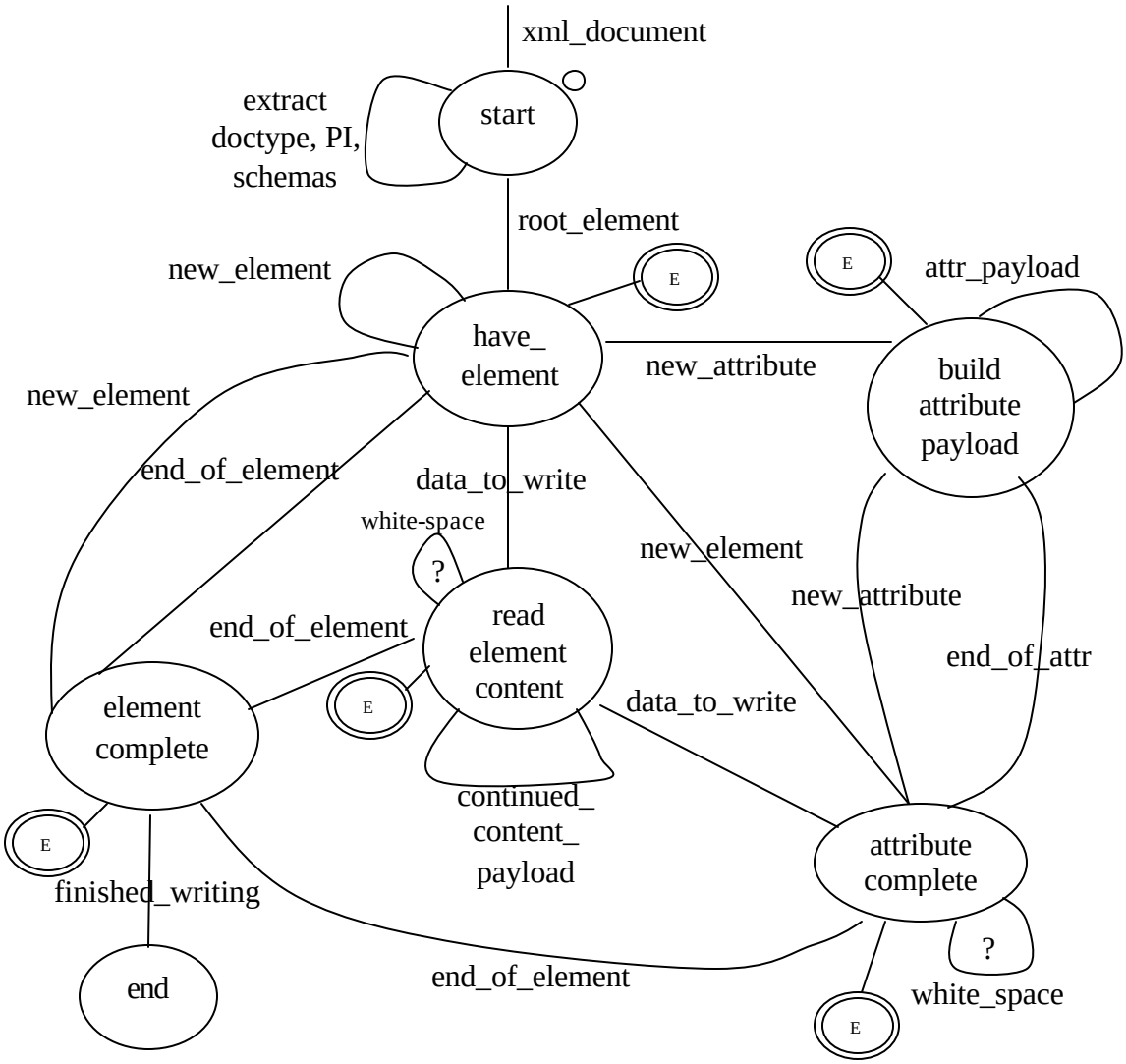

Figure 3.15: Communicating Finite State Machine (CFSM) Diagram of the Serialization Process

The state table for the CFSM used to validate and verify the serialization process is shown in Table 3.3. This table provides information about the purpose and actions taken in the states.

Name	Purpose / Action	Next Input / Transitions
start	Waits for the incoming XML Document. When XML document is passed in, the DOCTYPE, Processing Instruction and Schema information are removed.	Input : XML Document. Transition : The root element is passed to *"have element"* state.
have element	Waits for the root element of the document or the root element of the sub-elements. The element name is looked up from the table and the corresponding tag is put into the output stream.	Input : Start of the element. Transitions : 1- If data is present, state is transitioned to *"read content"* state via *data_to_write* transition. 2- If end_of_element is encountered, state is transitioned to "*element complete"* state. 3- If sub-elements are encountered, state is transitioned to itself via *new_element* transition. 4- If element start tag is not found in the look up table, an error is raised.

Name	Purpose / Action	Next Input / Transitions
read element content	Data Token “0” is put into the stream. Content from the XML Document is parsed and serialized according to its data types such as: 1- Non-continued data is put directly without putting size token. 2- If array type is encountered, first the data is parsed and its length measured; then the length token and data are put into the stream.	Input : Payload of an element. Whitespace is consumed as appropriate Transition : 1-If end of payload is encountered state is transitioned to *“element complete”* state. 2- If payload is a continued payload, state is transitioned to itself. 3- If the datatype of the corresponding element is not found an error is raised.
build attribute payload	Attribute Token “1” is put into the stream to signal the incoming attributes. This token is used only at the start of the first attribute. For attribute name the tag is looked up from the table and written to the stream. Data is parsed and handled as; 1. Non-continued data is put directly without putting size token. 2. If an array type is encountered, first the data is parsed and its length measured, the length token and data are put into the stream.	Input : Payload of an attribute. Transition: 1-If end of payload is encountered state is transitioned to *“attribute complete”* state. 2- If payload is a continued payload, state is transitioned to itself via the *attr_payload* transition. 3- If the datatype of the corresponding attribute is not found from the look up table, an error is raised.

Name	Purpose / Action	Next Input / Transitions
element complete	The end token for the element is looked up from the table and put into the stream.	Input : End of the element; next token Transition: 1-If the element to write is the last element in the XML Document, state is transitioned to state *"end"*. 2- If new element is encountered, state is transitioned to *"have element"* state via the *new_element* transition. 3- If the token corresponding to the element is not found from the look up table, an error is raised.
attribute complete	Checks for the new attributes, data for the current element, and new elements.	Input : End of the last attribute; next token Transition : 1-If all attributes are handled as indicated by receipt of end_of_element token, state is transitioned to *"element complete*" state. 2- If a new attribute token is encountered, state is again transitioned to *"build attribute payload*" state. 3- If token for payload data is encountered for the element, state is transitioned *"read element content"* state. 4- If new element token is encountered, state is transitioned to *"have element"* state. 5- If none of the above tokens are encountered, an exception is raised.

Name	Purpose / Action	Next Input / Transitions
E	Error Condition: declares the failure of the transition	In any state errors may occur. The reasons for the error state are listed below. 1 – Token numbers cannot be found for element or attribute. 2 - Data serialization cannot be handled properly. 3 - XML Document is not well-formed. 4- XML Document is not valid.
end	Declares the success of the serialization process.	Stream Serialization Complete

Table 3.3: Serialization Algorithm State Table

The transition table for XML Serialization is shown in Table 3.4. This table identifies the names, purposes, start and end states of the performed transitions in Serializer CFSM. The transitions are the changes from the beginning state to the end state and occur in the conditions stated in Table 3.3.

Name	Purpose	Start State	End State
extract doctype, PI, schemas	Extracts DOCTYPE, Processing Instruction and Schema Information from the XML Document prolog and root element	*start*	*start*
root_element	Transition from *start* to *have element*	*start*	*have element*

Name	Purpose	*Start State*	*End State*
new_element	Declares the existance of the new element	*have element*	*have element*
		attribute complete	
		element complete	
new_attribute	Declares the existence of first or new attribute	*have element*	*build attribute payload*
		attribute complete	
end_of_element	Declares the end of the element.	*have element*	*element complete*
		read content	
		attribute complete	
data_to_write	Declares that current element has data to write.	*have element*	*read content*
		attribute complete	
end_of_attr	Declares the end of the attribute is encountered	*build attribute payload*	*attribute complete*
attr_payload	The payload for this attribute is an array type.	*build attribute payload*	*build attribute payload*
continued content payload	The payload for this element is an array type.	*read content*	*read content*
finished_writing	Declares the end of the document is reached.	*element complete*	*end*
white_space	Removes white space between attributes or element tags as appropriate.	*attribute complete*	*attribute complete*
		read element content	*read element content*

Table 3.4: Serialization Algorithm Transition Table

In order to show how much bandwidth is saved, the previous text-based XML Document is sent in four different ways: first by sending the document as a text file; in the second and third, serializers JDOM and DOM4J are used, which remove whitespace; and in the last one XFSP is used. As seen in Figure 3.16 XFSP provides almost 60% compression over the other three methods for this example document.

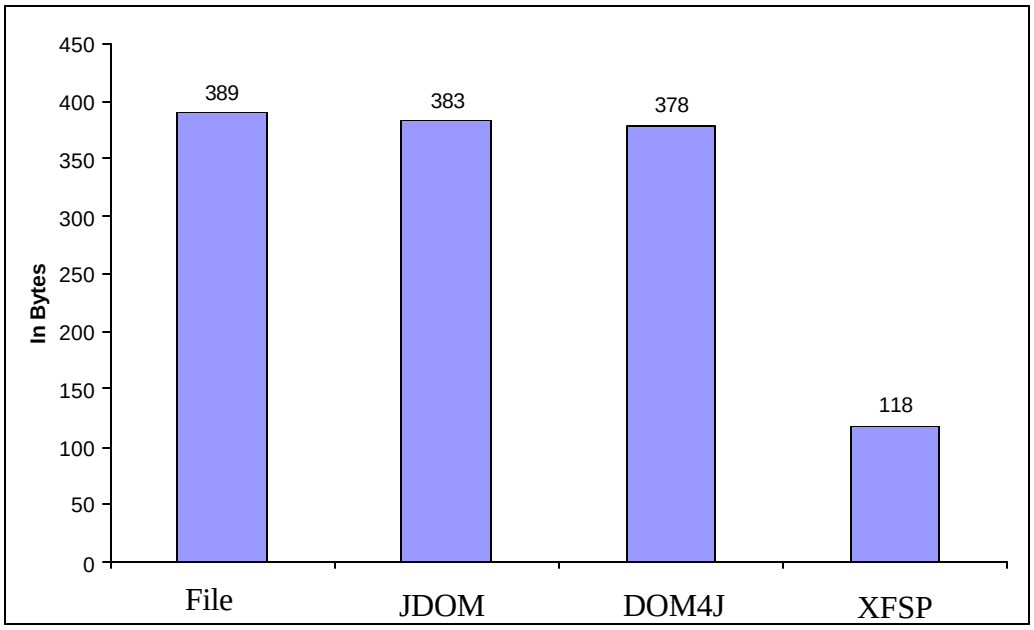

Figure 3.16: Comparison of Serialization Programs

F. XML DESERIALIZATION

XML Deserialization can be defined as recreating the XML tree back from the received binary stream. The purpose and implementation detail of sending XML documents in a compact way is described in the previous sections. In this section the process of deserializing back to the XML document is discussed.

For deserialization, *BinaryReader* class is implemented. The UML diagram for this class is shown in Figure 3.4. In order to construct a *BinaryReader* object, a look up table has to be created and passed as a parameter to the constructor of the class. That look up table is used for finding the element and attribute names and their data types as well. The pseudocode for binary reading operation is provided in Figure 3.17. In this operation, there can be two main parsing states, such as reading element state and reading attribute

state. These states signal the type of reading operation. If the parser is in reading element state then it can read a tag that corresponds to the start of an element, or a tag that corresponds to the end of the element, or a tag which states that actual data must be read next, or a tag which signals that a series of attributes will be read next.

When the parser enters the attribute-reading state, it can read a tag that corresponds to an attribute or a tag which signals the end of reading attributes which results with changing the reading state.

```
parse_state := element_reading;
current_element := null;

function binary_reader ( )
        do
                if (parse_state = element_reading)
                        read_elements( );
                if (parse_state = attribute_reading)
                        read_attributes ( );
        while ( not_end_of_stream)
end function;

function read_elements ()
        read_tag();
        if (element_start_tag)
                create_element();
                bind_to_current_element();
                push_to_stack();
        else if (element_end_tag)
                current_element := pop_stack();
        else if (data_tag)
                read_data();
                bind_to_current_element();
        else if (attribute_start_tag)
                parse_state := attribute_reading;
end function;

function read_attributes ()
        read_tag();
        if (attribute_end_tag)
                parse_state : = element_reading;
        else
                read_attribute_tag ();
                create_attribute();
                read_data();
                bind_to_current_element();
end function;
```

Figure 3.17: Binary Reader Pseudocode

As seen in pseudocode, the main approach in recreating the XML document is gradually building the tree back by using stack operations. The major steps in this process can be seen in Figure 3.18, some in-between steps are skipped to reduce the complexity in the figure.

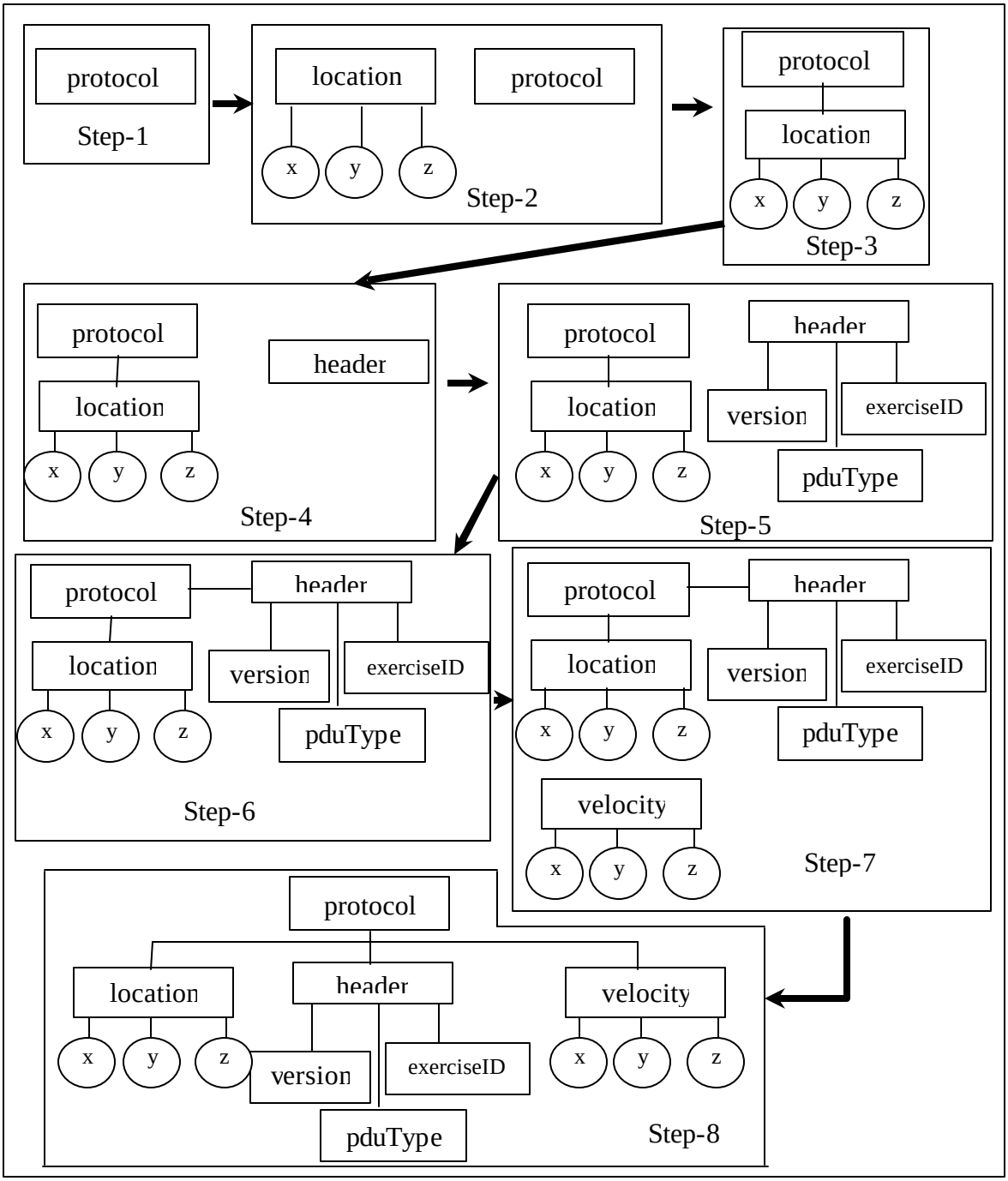

Figure 3.18: XML Deserialization

The Deserializer CFSM is shown in Figure 3.19. This figure defines the algorithm used to deserialize the received stream back into the original XML tree.

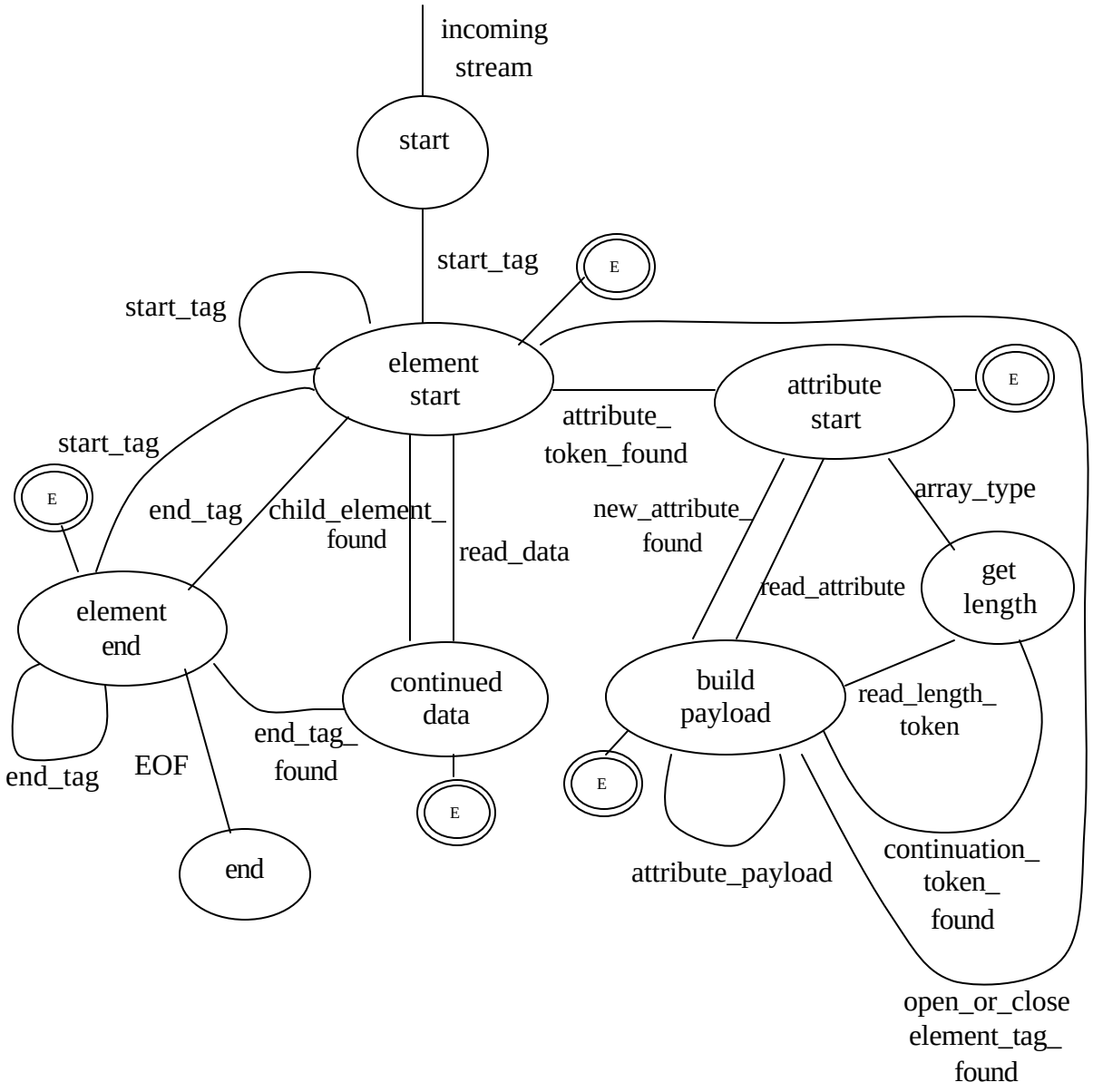

Figure 3.19: Communicating Finite State Machine (CFSM) Diagram of Deserialization Process

The state table for the CFSM used to validate and verify the deserialization process is shown in Table 3.5. This table provides information about the purpose and actions taken in the states.

Name	Purpose / Action	Input / Transitions
start	Waits for the incoming stream.	Input : Incoming serialized stream Transition : When the tag for the root element is read and looked from the table, state is transitioned to *"element start"* state.
element start	The element for the deserialized tag is created and pushed to the stack.	Input : Token for the name of the element Transition : 1- If another tag for a new element is encountered, state is transitioned to itself. 2- If end tag for the current element is encountered, state is transitioned to "*element end*" state. 3- When data token is encountered, state is transitioned to *"continued data*" state. 4- In case that an attribute start token is found, state is transitioned to *"attribute start*" state. 5- When none of the above tokens are found, an exception is raised.

Name	Purpose / Action	Input / Transitions
attribute start	The start tag for the attribute names is parsed from the stream, then a new attribute is created and bound to the current element.	Input : The special token "1" which indicates the start of the attributes. Transitions : 1-If data type of the current attribute is not variable length, state is transitioned "*build payload*" state via *read_attribute* transition. 2- If data type of the current element is variable length, the state is transitioned to "*get length*" state via *array_type* transition. 3- When none of the above tokens are found, an exception is raised.
build payload	Builds the payload of the current attribute and binds the created data object to the attribute.	Input : The payload of the attribute Transitions : 1- If start of new attribute is encountered, state is transitioned to "*attribute start*" state. 2- If open or close tag of the element is found, state is transitioned to "*element start*" state. 3- In variable length data case, payload of the current attribute is created by reading special tags between data blocks. 4- If continuation token is found, state is transitioned to "*get length*" state. 5- When none of the above tokens are found, an exception is raised.

Name	Purpose / Action	Input / Transitions
continued data	Builds the payload of the current element and binds it to itself.	Input : The data to parse Transitions : 1- If child element is found, state is transitioned to *"element start"* state. 2- If end of the current element is encountered, state is transitioned to "*element end*" state. 3- If data cannot be parsed or when none of the above is found, an exception is raised.
element end	Pops the element from the stack and binds it to current root element.	Input : End tag of the current element. Transitions : 1- If a start tag for a new element is encountered, then state is transitioned to "*element start*" state. 2- If end of the current stream is found, then state is transitioned to "*end*" state. 3- If another end tag of another element is found then state is transitioned to itself. 4. When none of the above tokens are found, an exception is raised.

Name	Purpose / Action	Input / Transitions
get length	Reads the length of the variable length data.	Input : The data type of the current attribute or data continuation token. Transitions : 1- The length of the data is read and state is transitioned to *"built_payload"* state.
E	Error condition: declares the failure of the transition	In any state errors can occur. The reasons for the error state are listed below. 1 - Tag numbers cannot be found. 2 - Data deserialization cannot be handled properly. 3 – Invalid tokens are received.
End	Declares the success of the deserialization process.	Input : End of the data stream Transitions : --

Table 3.5: Deserialization Algorithm State Table

The transition table for XML Deserialization is shown in Table 3.6. This table identifies the names, purposes, start and end states of the performed transitions in the CFSM.

Name	Purpose	Start Sate	End State
incoming stream	The start of the process is declared.	---	*start*

Name	Purpose	*Start State*	*End State*
start_tag	The start tag of an element is read.	*start*	*element start*
		element start	
		element end	
end_tag	The end tag of the element is read.	*element start*	*element end*
		element end	
		continued data	
read_data	Data is encountered	*element start*	*continued data*
attribute token_found	Special token “1” is read from the stream	*element start*	*attribute start*
child_element_found	Another start tag is found during the processing current element.	*continued data*	*element start*
read attribute	The token for the current attribute is read.	*attribute start*	*built payload*
new_attribute_found	A new attribute is found (token for the new attribute is read from the stream).	*build payload*	*attribute start*
attribute_payload	Continuation of parsing the payload of the current attribute.	*build payload*	*build payload*
read_length_token	The length token for the variable length attribute payload is read.	*get length*	*build payload*

Name	Purpose	*Start State*	*End State*
continuation_token found	A special token for payload continuation is found.	*build payload*	*get length*
open_or_close element_tag found	Open or close tag for an element is read.	*built payload*	*element start*
array_type	If data type of the current attribute is an array type, where the data type is looked up from the table.	*attribute start*	*get length*
EOF	End of stream is read.	*element end*	*end*

Table 3.6: Deserialization Algorithm Transition Table

G. DATA TYPES

In XFSP 13 native data types [W3C Schema Part II] are implemented. These data types provide the foundation of data structures that XFSP can understand. The main point of implementing XFSP data types is pushing the serialization and deserialization of data into the classes where data is actually stored.

These data objects are created when the text-based XML document is parsed or when a serialized stream is received at the user end. The UML diagram for implemented data types is shown in Figure 3.5.

The data types can be divided into two main categories such as Complex Type and Simple Type. Complex Type provides name storage for the XML elements where they implement non-primitive type data structures. Simple Types are the objects where actual primitive or array type data is stored. These data types have the capability to store the data, serialize it into the given output stream, deserialize it from the received input stream and provide that data as text to the user. The implemented data types are shown below. The prefix XSD is used to distinguish these types from the primitive types used in the Java programming language.

- XSDByte
- XSDUnsignedByte
- XSDShort
- XSDUnsignedShort
- XSDInteger
- XSDUnsignedInt
- XSDLong
- XSDFloat
- XSDDouble
- XSDBoolean
- XSDString
- XSDComment
- SimpleTypeExtension

In XFSP, these objects are bound to the XML tree during the tree generation process. These objects can represent primitive data structures as well as user defined complex data types.

H. XFSP AND NPSNET-V

XFSP is introduced as a run-time extensible application layer protocol into the NPSNET-V. As discussed before, NPSNET-V is a run-time extensible networked virtual environment architecture having both DIS and HLA capabilities to share information for participating users. The major advantage of using NPSNET-V to show the run-time extensibility of application protocols is its component based framework allowing components to be loaded at run-time to build the actual virtual-environment architecture.

NPSNET-V uses many well-known design patterns. The major ones that are used in XFSP components are *Interface* and *Listener* patterns. Both of these design patterns are highly used in the XFSP component in NPSNET-V architecture.

The idea behind the Interface pattern is "… Keep a class that uses data and services provided by instances of other classes independent of those classes by having it access those instances through an interface" [Grand 98]. This pattern allows for a plug-and-play type of architecture within an application. "For example a certain application relies on a specific service to be provided, and that service can be provided by different service providers. All that the application interface needs to specify is a set of functions that any service provider must implement in order to provide the desired service. Any service provider that is to be used with that application must implement the function required by the interface in order for the applications to be able to use its services" [Salles 02].

The second pattern is the Listener pattern, also commonly known as *Observer* pattern. This pattern lays the foundation for efficient notification of events occurring in one object to be transmitted to registered objects.

In order to implement the XFSP component in NPSNET-V, two main controller modules, *StandardXFSPController* and *XFSPExplosionManagerController* are implemented. These modules can understand entity state and explosion packets defined by XML-Schemas.

These controllers define the semantics between the protocol syntax and application. They cause action of the received packets to be represented correctly in a 3D world when they are issued by the participating entities. Currently the semantics of fire and collision packets are not defined in the NPSNET-V framework, but XFSP can parse those packets and create the XML documents from the received streams.

UML diagram of classes implemented to integrate XFSP and NPSNET-V is shown in Figure 3.20.

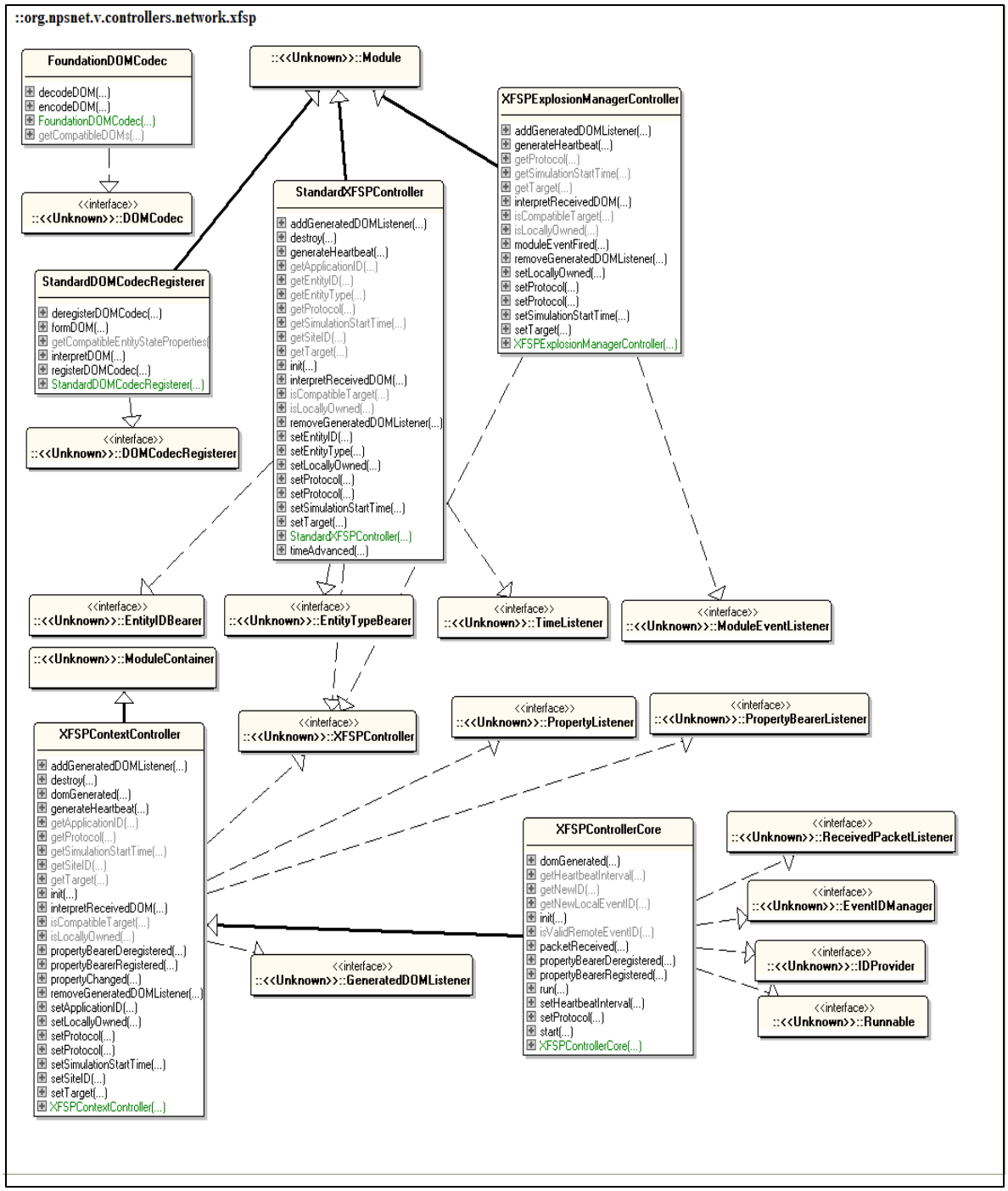

Figure 3.20: StandardXFSPController UML Diagram (Generated by [ESS])

The following pictures show the run-time extensibility of the protocols in NPSNET-V. Figure 3.21 shows the start of the application with two ships communicating via multicast where they implement the same entity state protocol. Each ship is controlled by separate processes running on the same computer or separate hosts. In the presented pictures, therefore, one ship is local controller (master) and the other is ghosted copy (slave).

Figure 3.21: Start of NPSNET-V Application

Figure 3.22: Run-time Detonation Protocol Loading

After the start of application, one of the clients loads a detonation entity associated with a detonation protocol schema. At the parsing process the entity that have loaded the schema sends a special message to the multicast group declaring the new protocol that it loaded. When the other entities receive the special message they parse the schema pointed to by the URL in the message and are then able to understand the detonation protocol syntax. Figure 3.22 shows one of the clients that received the message and parsed the schema pointed to in that message, resulting in execution of a detonation behavior. The format of the special message is shown below in Figure 3.23.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with XML Spy v4.3 by Ekrem Serin (NPS) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="xfsp">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="manager" type="managerType"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="managerType">
        <xs:sequence>
            <xs:element name="entitySite" type="xs:short"/>
            <xs:element name="entityApplication" type="xs:short"/>
            <xs:element name="entityID" type="xs:short"/>
            <xs:element name="schemaURL" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Figure 3.23: NPSNET-V Simulation Manager Type Packet Format

I. SUMMARY

This chapter provides information about the implementation details of XFSP and an exemplar usage in a Net-VE. Schema parsing, XML Serialization and XML Deserialization processes are covered in detail to address the research question and show how an XML document can be compressed and restored.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. BINARY X3D

A. INTRODUCTION

This chapter examines the design and implementation details of a binary X3D format for network streaming and file storage.

B. OVERVIEW

Extensible 3D (X3D) is an XML file format that describes a scene graph which is rendered as a 3D scene. X3D scene graph is a directed acyclic graph with nodes and edges. Nodes in the scene graph define the graphic and aural objects contained in the system, and the edges define the transformation hierarchy containing the spatial relationship of objects [Web3D]. X3D is the third-generation version of the ISO standard for The Virtual R-M-C-VRML97.

Being an XML text file format makes X3D fairly heavyweight for network transmission as well as storage purposes. Text files generally require much more bandwidth than their binary equivalents. The compressed versions of text files offer an efficient way to send or receive data over the network as well as to store them locally. The idea of schema parsing, XML Serialization and XML Deserialization is used to compress the X3D files. The output file format is called "*Binary X3D*" with `.b3d` or `.b3z` file extensions depending on whether or not GZIP compression is further used.

The following sections describe and analyze the binary X3D program by providing essential details on process flow.

C. X3D-EDIT

X3D-Edit is an authoring tool for X3D graphic scenes developed by using IBM's Xeena, an XML-based tool-building application. It is a graphics file editor for X3D files that enables simple error-free editing, authoring and validation of X3D scene graph files [Brutzman X3D]. X3D scene graph files are translated to VRML97 syntax by using an XSLT stylesheet. The converted file format can be rendered by open-source browsers

(Xj3D) as well as commercial products; e.g., Internet Explorer by providing a plug-in. Figure 4.1 shows the X3D Edit interface for a typical *"Teapot"* example.

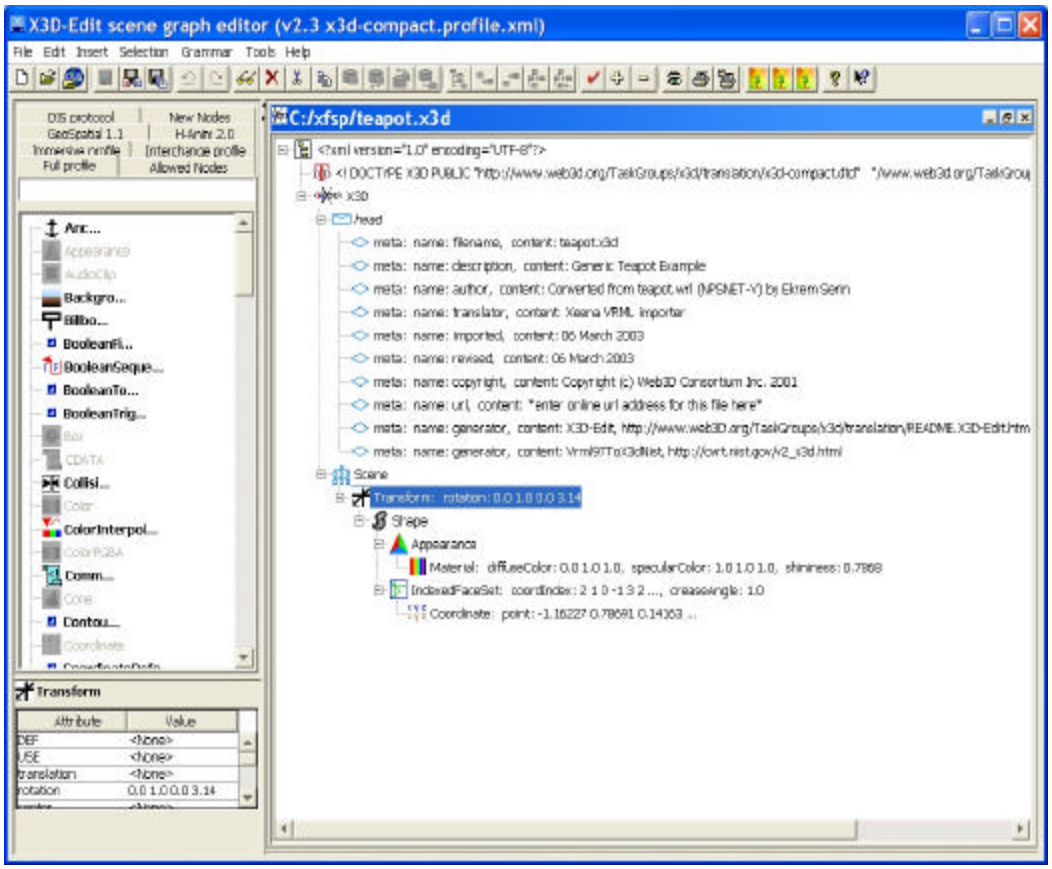

Figure 4.1: Teapot.x3d File Example

Figure 4.2 shows the output file rendered by using Cortona [Cortona 03] plug-in for Internet Explorer browser. The rendered file is in VRML97 file format and generated by applying an XSLT stylesheet to the teapot.x3d file. The Figure 4.1 and Figure 4.2 are presented to show the how an XML file that defines a 3D scene graph can be rendered.

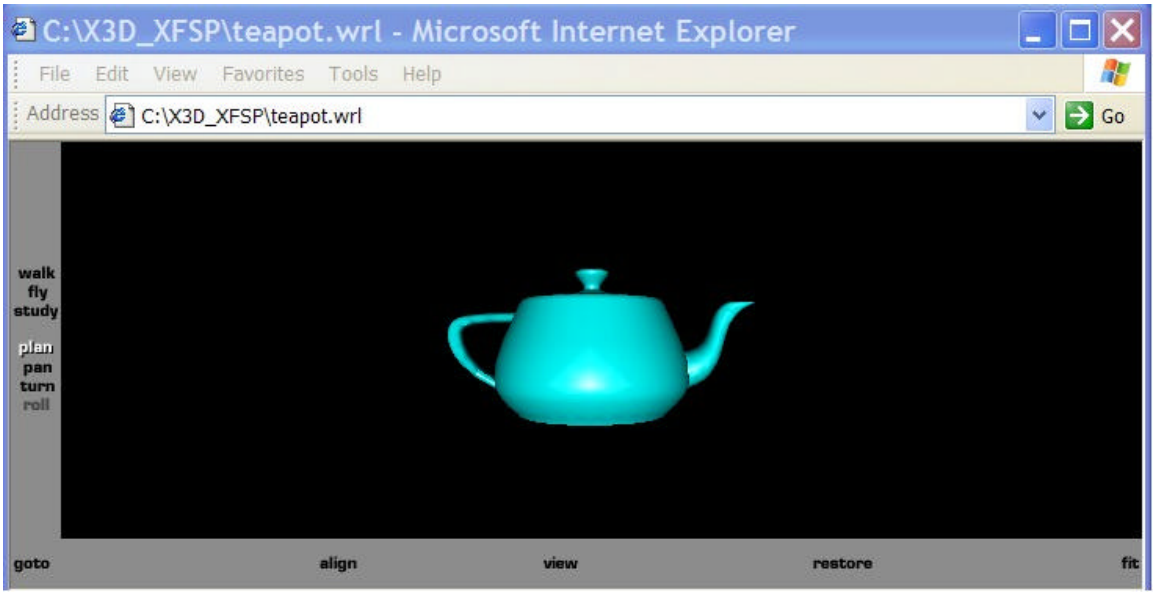

Figure 4.2: Teapot.wrl File Example

D. BINARY X3D

1. Binary X3D Program Interface

Binary X3D program provides a compact way to store X3D files by exploiting the idea of schema parsing and XML Serialization described in previous chapter. In addition to XML Serialization, binary X3D program provides options to use GZIP streams to further compress the X3D files. The interface for the binary X3D program is shown in Figure 4.3. The implementation details and process flows are discussed at each option description.

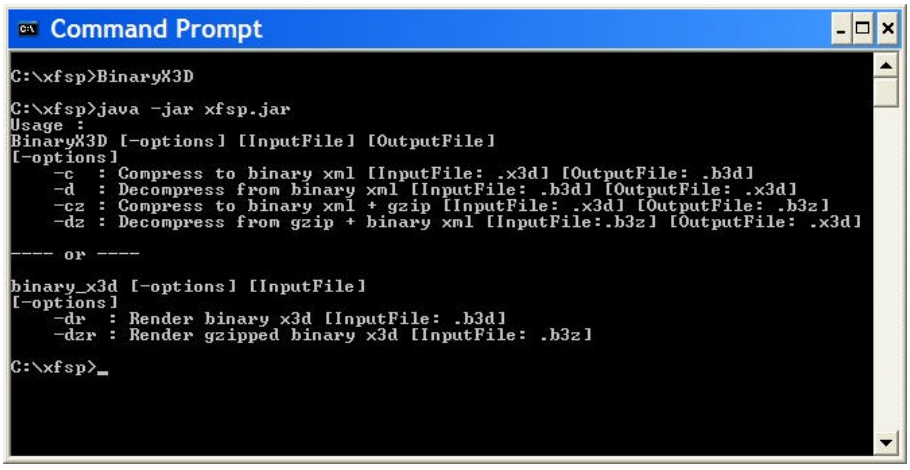

Figure 4.3: BinaryX3D Program Interface

- The option “-c” is used to compress the X3D file to a binary X3D file. The output file extension is considered as `.b3d` which stands for binary X3D. The process for this operation is shown in Figure 4.4. The X3D file is provided as parameter to the Serializer program and compressed by using X3D Schema.

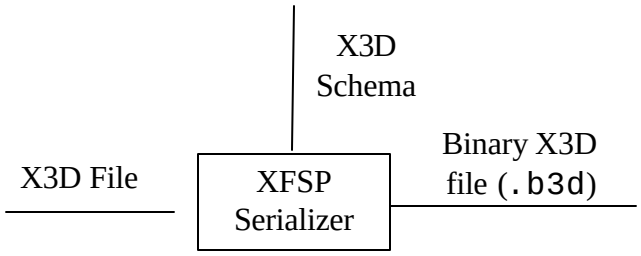

Figure 4.4: Binary X3D File Generation

- The option “-d” is used to decompress binary X3D file format. The process for this operation is shown in Figure 4.5. The `.b3d` (Binary X3D) file is provided as a parameter to the Deserializer and decompressed by using X3D Schema.

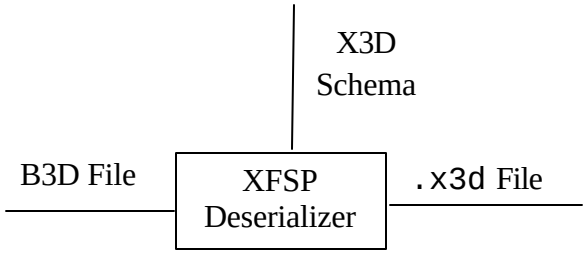

Figure 4.5: `.x3d` File Generation from `.b3d` File

- The option “-cz” is used to compress the X3D file by using Serializer program and GZIP streams. The output file format is considered as B3Z which stands for gzipped binary X3D. The process for this operation is shown in Figure 4.6. The X3D file is provided as a parameter to the Serializer program where it is serialized using X3D Schema and compressed using GZIP compression.

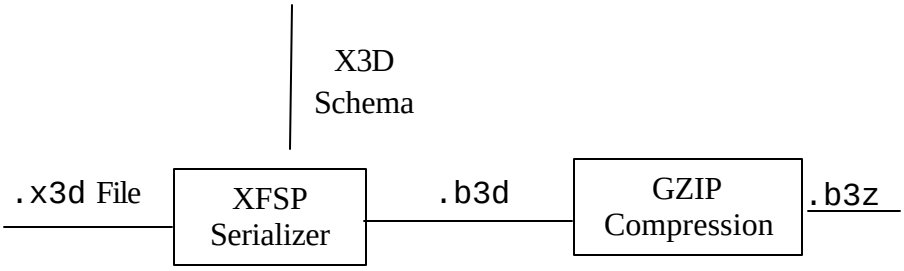

Figure 4.6: .b3z File Generation (Gzipped Binary X3D)

- The option "-dz" is used to decompress gzipped binary X3D (.b3z) file back to the .x3d file format. The process for this operation is shown in Figure 4.7. The .b3z file is provided as a parameter to the GZIP stream and decompressed. The decompressed file (.b3d) is provided to Deserializer program and converted back to the .x3d file by using X3D Schema.

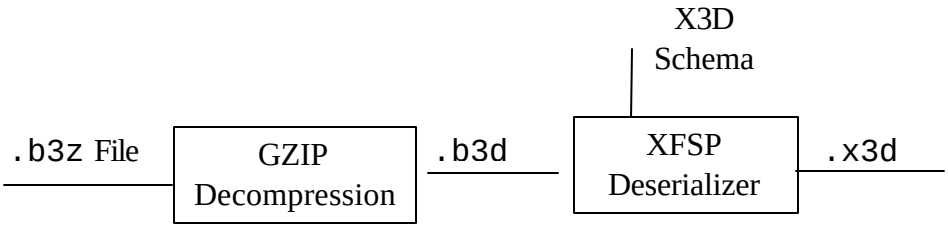

Figure 4.7: .x3d File Generation from .b3z File

- The option "-dr" is used to decompress binary X3D (.b3d) files and render them. To render .b3d file format an XSLT stylesheet is applied to the decompressed B3D (.x3d) file and transformed to VRML97 file format. The VRML97 file format is loaded into a VRML or X3D browser (e.g. Xj3D). The complete process flow for this operation is shown in Figure 4.8.

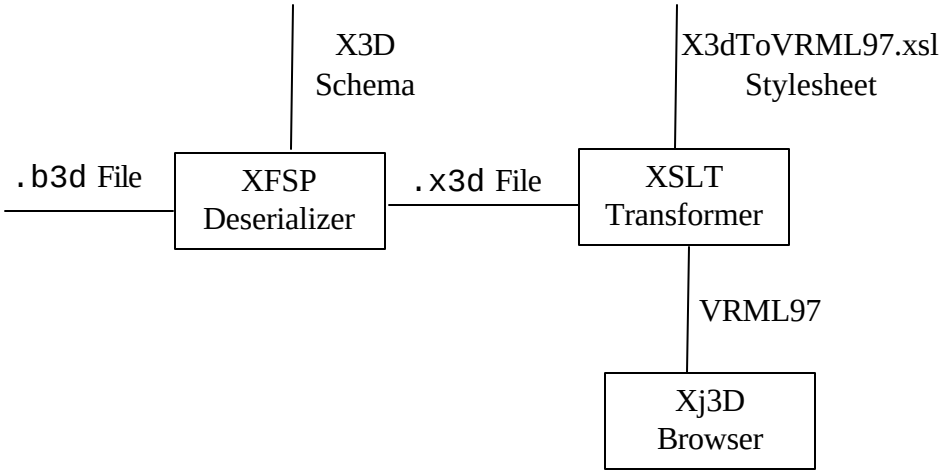

Figure 4.8 : Rendering `.b3d` File Format

- The option "-dzr" is used to decompress `.b3z` files and render them by using a browser. To render `.b3z` file format, the input file is decompressed by gzip stream and converted to a `.b3d` file format. The `.b3d` file is converted to `.x3d` file by using the XFSP Deserializer program. An XSLT stylesheet is applied to the X3D file and transformed to VRML97 file format. The VRML97 file is rendered by using Xj3D browser. The complete process flow for this operation is shown in Figure 4.9.

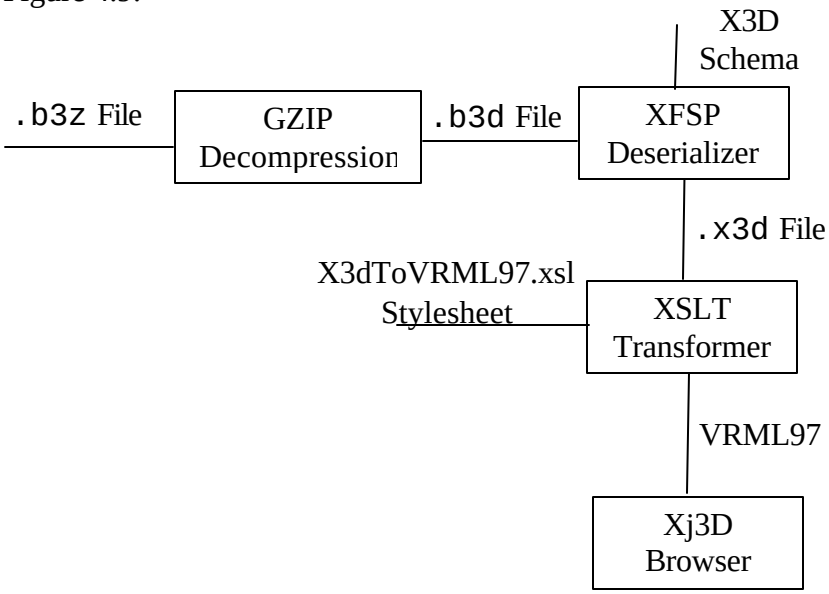

Figure 4.9: Rendering `.b3z` File Format

2. Analysis

To test the designed program, a common example *"Teapot"* is used. The *teapot.x3d* file contains 14994 integer numbers that define the coordinate indices, and 5916 floating point numbers that represent the coordinates of vertices in 3D. Furthermore, the *Teapot* file includes various elements and attributes that define the material and color properties of the object. The rendered teapot is shown in Figure 4.2.

The following charts shows the data collected by using different file formats and compression schemes. In Figure 4.10 the x-values represent the file format, and y-values represent the size of the file in Kbytes. The file formats used for comparison are described below.

- `x3d` : X3D File Format
- `wrl` : VRML97 File Format
- `b3d` : Binary X3D File Format
- `zip_x3d` : X3D File Compressed by WinZip Program
- `zip_wrl` : VRML97 File Format Compressed by WinZip Program
- `b3z` : X3D File Compressed by Serializer using GZIP Streams

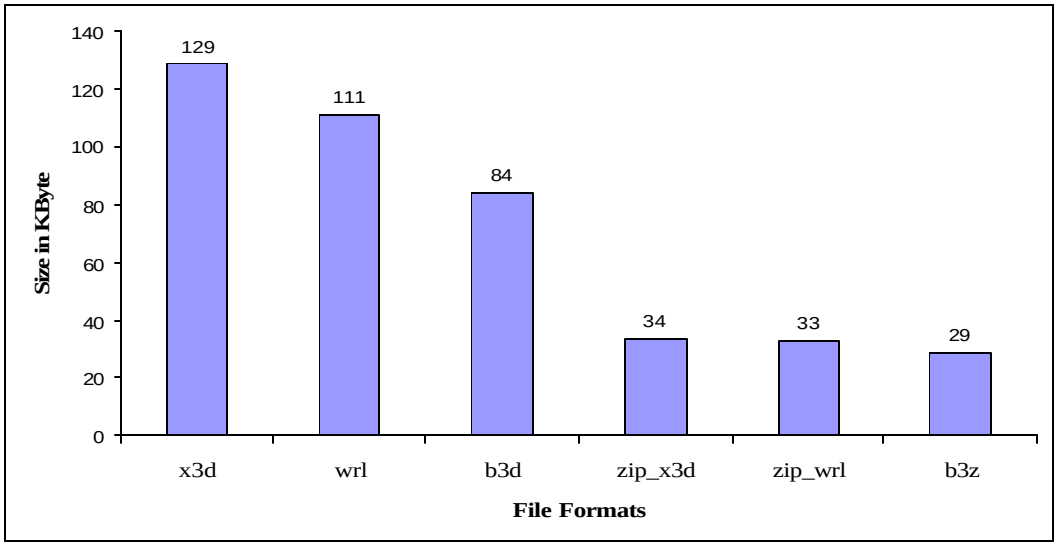

Figure 4.10: File Format Comparison for Teapot Exemplar

Figure 4.11 represents the storage and bandwidth gained by using different file formats. The base for percentage calculation is set as the teapot.x3d file. The B3Z file format provides 78% bandwidth saving over the regular X3D file format. Furthermore, B3Z file format provides 12% saving over the compressed WRL file format commonly known as WRZ file format. The *teapot.x3d* example contains much more data than the element and attributes names. For an exemplar including many elements and attributes, the storage savings will increase due to the Serializer program compression. In that case, attribute and element names will need more storage and the Serializer program will work more effectively to compress the X3D file.

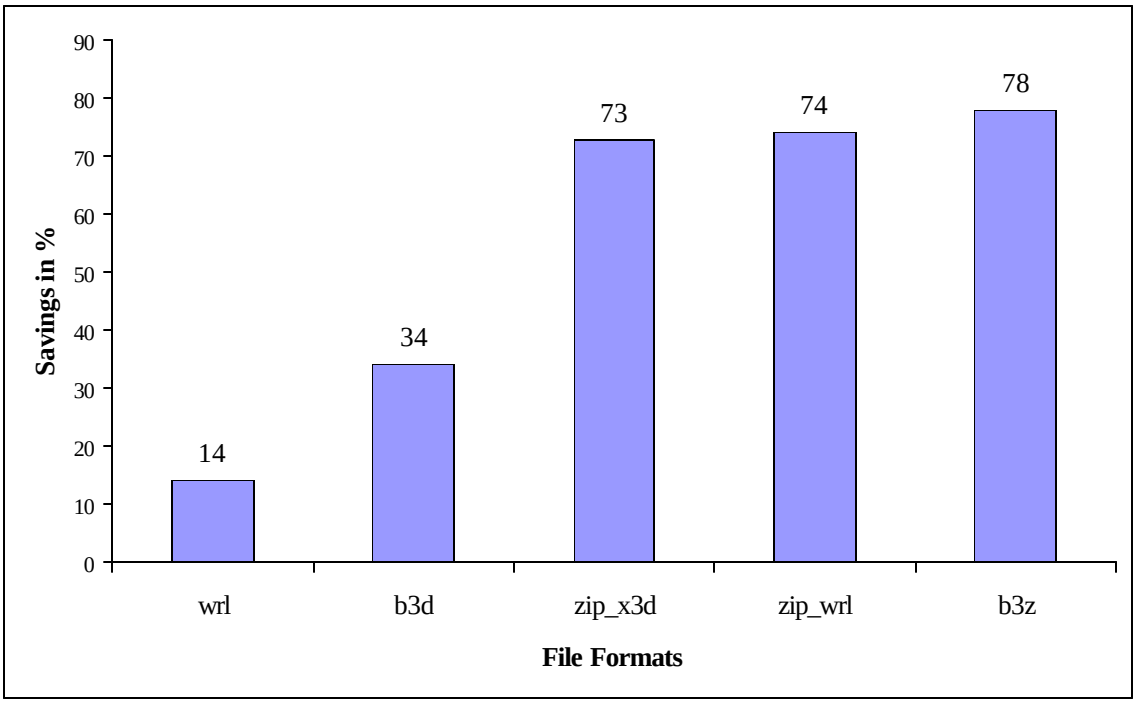

Figure 4.11: File Format Percentage Saving for Teapot Exemplar

The properly reconstructed and rendered *teapot.b3z* file is shown in Figure 4.12. The total amount of time for decompression, stylesheet transformation and rendering is measured as 5 seconds. The experiment was conducted on a Dell Inspiron 8200 Laptop with P4 1.6GHz CPU, 512 MByte RAM and 64 MByte GeForce-4 graphics card.

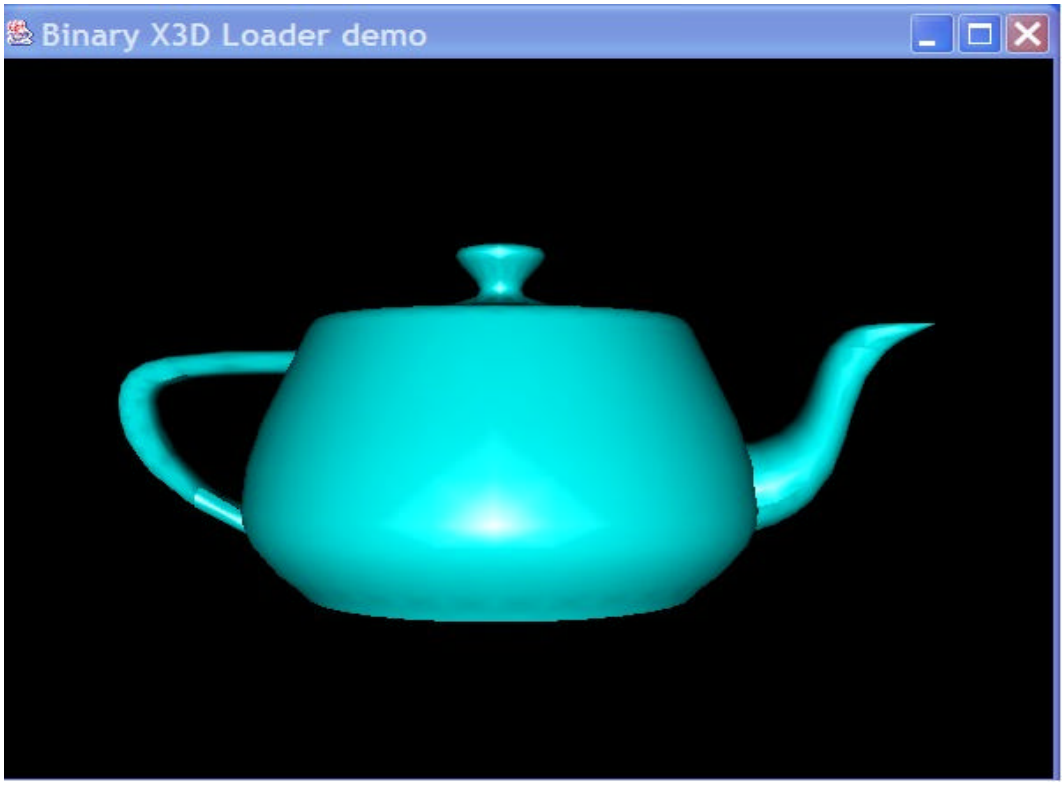

Figure 4.12: Rendered *teapot.b3z* File

E. SUMMARY

This chapter provides information about the implementation details of binary X3D file formats (`.b3d` and `.b3z`). `.b3d` file format offers 34% bandwidth gains over regular X3D file format; B3Z file format offers 78% bandwidth gains for Teapot exemplar. Additionally, `.b3z` file format provides 12% bandwidth gains over zipped VRML97 file format. These compression improvements from using XML are consistent with other results. The implemented program provides an effective way to use network bandwidth as well as local storage capacity by using XML Serialization and GZIP stream compressions.

THIS PAGE INTENTIONALLY LEFT BLANK

V. PROTOCOL DATAGRAM UNIT (PDU) FARM

A. INTRODUCTION

This chapter examines the design and implementation details of *Pdu Server*, *Pdu Capture* and *Network Analyzer* programs.

B. OVERVIEW

In order to test the designed Net-VE for 24 hours a day and 7 days a week, three main programs, *Pdu Server*, *Pdu Capture* and *Network Analyzer* are implemented. As their names imply, these programs are used to send and receive packets as well as to monitor the network state between end points.

In the monitoring process, previously described metrics such as latency, drop-rate and jitter are collected and stored in a file to help users draw conclusions about the current state of the network between the end points.

Protocol Datagram Unit (PDU) Farm is defined as the collection of computers where previously recorded or run-time generated packets are continuously transmitted over the network. In order to establish a PDU Farm, implemented programs are loaded to four different computers and run continuously. Current implementation of PDU Farm needs previously recorded packets to execute its task. The main point is mimicking a previously played scenario to draw conclusions about different metrics such as network state, rendering performance or memory usage for Net-VEs.

In the benchmark tests the total bandwidth of four computers is measured as 1.6 Mbps. With this bandwidth 300 different entities with one packet per second send rate can be represented in highest resolution XFSP (670 bytes) packet format.

The implementation details of PDU Farm are covered in the following sections. In order to present the design scheme of PDU Farm, a UML diagram is provided in Figure 5.1. The UML diagram presents the implemented Java classes.

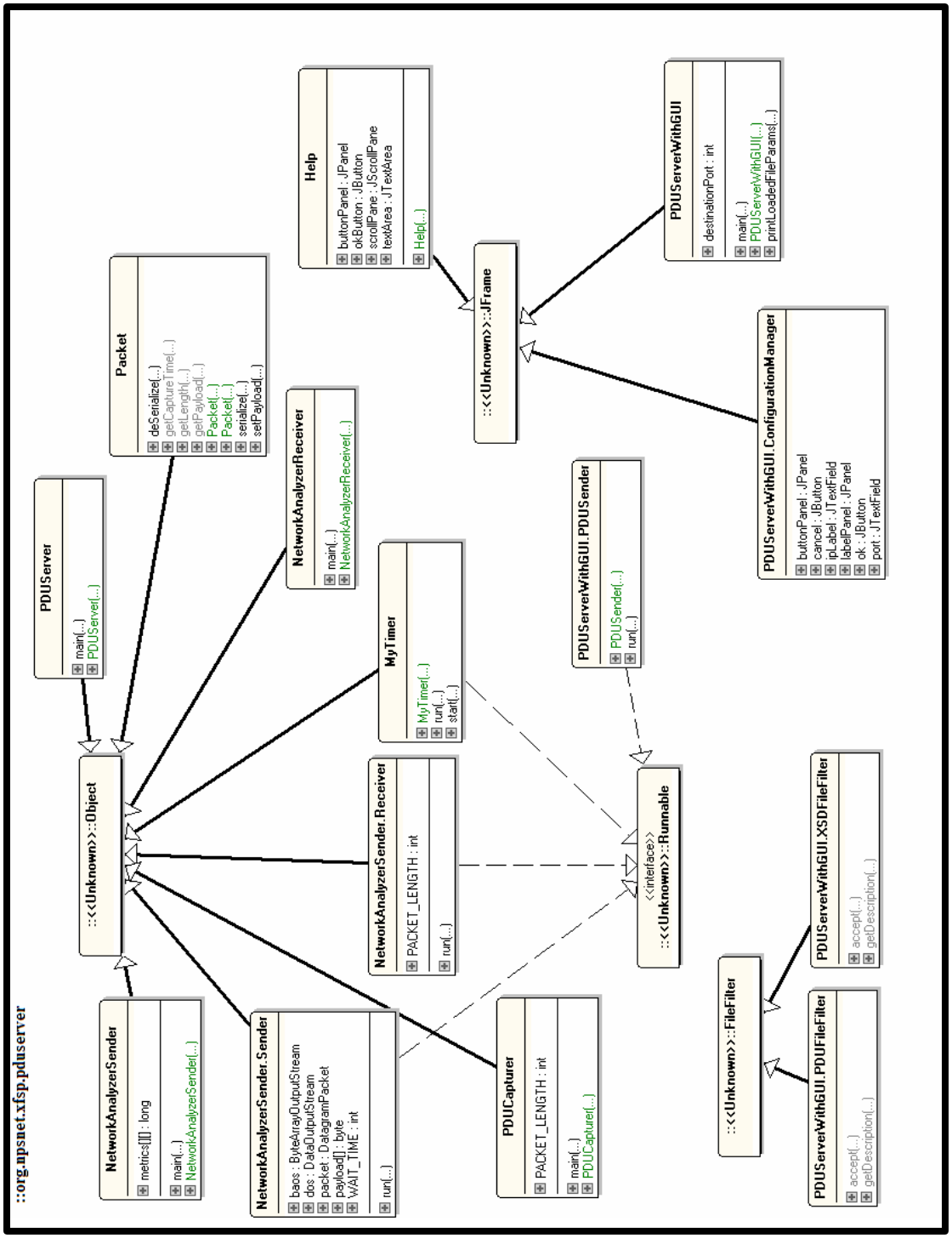

Figure 5.1: PDU Farm UML Diagram (Generated by [ESS])

C. PDU SERVER

The *Pdu Server* program is used to send previously recorded packets to the end-users via unicast or multicast transport. These previously recorded packets are stored in a file where they are encapsulated by a special wrapper. This wrapper is used to determine the capture time and capture order of the received packet.

In order to simulate the previously recorded scenario precisely, the captured packets are sent at the rate and in the order of their capture intervals. These time intervals define the send time of the captured packet.

Starting the *Pdu Server* is done by a batch (DOS) or bash (Linux) file. This file uses an XML file to define initialization parameters of the server. An example initialization file is shown in Figure 5.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with XML Spy v4.3 by Ekrem Serin (NPS) -->
<System xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<!-- Time pattern is mm.dd.yyyy hh:mm:ss-->
    <StartTime>03.29.2003 09:34:00</StartTime>
    <EndTime>03.29.2003 11:35:00</EndTime>
    <Loop>true</Loop>
    <SocketIP>225.23.93.25</SocketIP>
    <Port>61000</Port>
    <URL>file://c:/xfsp/nps.pdu</URL>
</System>
```

Figure 5.2: *Pdu Server* Initialization Parameters

Start and end time fields in Figure 5.2 define the start and end time of the application. In the given example the server will start running at 29th of March 2003 at 9:34:00 am. and will stop at 29th of March 2003 at 11:35:00 am. Between the actual start time and the start of the application, the server will enter to sleep state. At the end of the sleep state the server will wake up and will start sending previously recorded packets.

The Loop field in this initialization document defines whether the server will use the same file where previously recorded packets are stored continuously between its start and end. The Loop *true* means that the server is going to use the same file continuously

during its operation. When the server reaches the end of the file it will restart sending the packets from the top of the file.

The SocketIP field determines the IP address that the server will use to send the recorded packets. The server is capable of sending both unicast as well as multicast packets. If the provided IP is in the multicast IP address range (224.0.0.0 - 239.255.255.255) the server will open a multicast socket and join that address group. In the unicast IP case, the server will open a unicast socket and send the packets to that IP address.

The Port field is used to define the port number to send the packets. A port number is an integer number between 1- 65535 and differentiates the applications running on the same computer with the same IP. It can be considered as the multiplexer where the operating system uses it to hand the incoming data to the correct application.

The URL (Uniform Resource Locator) field determines the location of the file that will be used for sending the packets. As mentioned before, that file stores the previously recorded packets.

When the initialization parameters are provided with the XML document, the server uses these parameters to set its internal attributes. After setting internal attributes, the server sends the recorded packets to a unicast user or to a multicast group. The output of the PDU Server program is shown in Figure 5.3. In order to run the server remotely on computers with Linux Operating System (OS), no graphical user interface is used. Text-based command-line startup simplifies remote invocation.

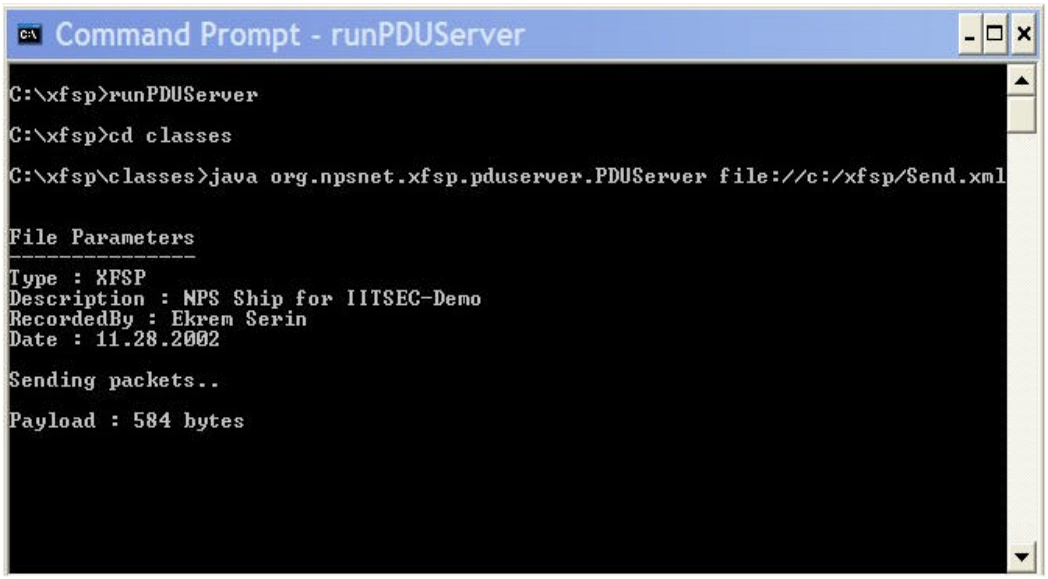

Figure 5.3 : Output of PDU Server Program

D. PDU CAPTURER

The *Pdu Capture* program is used to capture the packets generated by other users and store them in a file for the future use to regenerate the previously played scenario. The server listens to a unicast or multicast socket and receives the packets. At each packet receive interval it wraps the packet payload by a special wrapper where it puts the length and receive time of the received packet in that wrapper.

Starting of *Pdu Capture* program is similar to *Pdu Server* where a batch (DOS) or bash (Linux) file is used. This file uses an XML file to define initialization parameters of the capturer program. An example initialization file is shown in Figure 5.4.

Start and end time fields in Figure 5.3 are similar to PDU Server program. These fields define the start and end time of the application. In the given example the server will start running at 29th of March 2003 at 4:55:00 pm. and will stop at 29th of March 2003 at 6:15:00 pm. Between the actual start time and the start of the application, the capturer will enter to sleep state. At the end of the sleep state, the capturer will wake up and will start capturing the packets.

```
<System xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<!-- Edited with XML Spy v4.3 by Ekrem Serin (NPS) -->
    <!-- Time pattern is mm.dd.yyyy hh:mm:ss-->
    <StartTime>03.29.2003 16:55:00</StartTime>
    <EndTime>03.29.2002 18:15:00</EndTime>
    <SocketIP>127.0.0.1</SocketIP>
    <Port>61000</Port>
    <HeaderFileURL>file://c:/xfsp/FileHeader.xml</HeaderFileURL>
</System>
```

Figure 5.4: *Pdu Capture* Program Initialization Parameters

The SocketIP field determines the IP address that the *Pdu Capture* program will use to receive the packets. The server is capable of receiving both unicast as well as multicast packets. If the provided IP is in the multicast IP address range (224.0.0.0 - 239.255.255.255) the capturer will open multicast socket and join that address group. In the unicast IP case, the capturer will open a unicast socket and receive the packets on that IP. If the provided unicast IP address is not the local-host or not the IP address of one of its network interfaces then the capturer will not be able to receive any incoming packets.

The Port field is used to define the port number to receive the packets. A detailed description port number is provided in PDU Server program section.

The HeaderFileURL field determines the location of the header file, an XML file, which is copied to the top of the file where the captured packets are stored. The reason for using an approach like this is to be able to see the content of the binary file in which the captured packets are stored. An example of the header file is shown in Figure 5.5. When the recorded PDU file is opened by notepad or any text file viewer the XML file in Figure 5.5 will be on top of that file where it describes the content of the opened file. The captured packets will follow this XML-text in binary format where they will not be correctly interpreted by the text file viewer used. This provides the user the flexibility of seeing at least the content of a binary file with a text file viewer.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with XML Spy v4.3 by Ekrem Serin (NPS) -->
<FileHeader xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Type>XFSP</Type>
    <Description>NPS Ship for IITSEC-Demo</Description>
    <RecordedBy>Ekrem Serin</RecordedBy>
    <Date>11.29.2002</Date>
    <URL>file://c:/xfsp/xfspNPS.pdu</URL>
</FileHeader>
```

Figure 5.5: A File Header for Recorded PDU File

The output of the PDU Capturer program is shown in Figure 5.6. In this output, the program prints the initialization parameters and starts listening for the incoming packets.

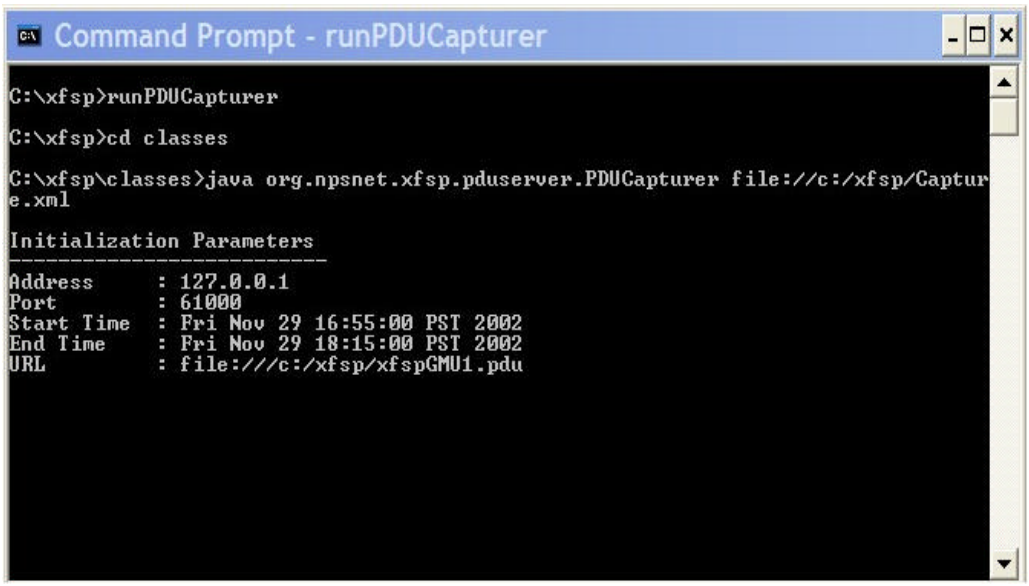

Figure 5.6: Output of *Pdu Capture* Program

E. NETWORK ANALYZER

The *NetworkAnalyzer* program is used to collect the previously decided metrics such as latency, drop-rate and jitter between two end-points. With this feature users are able to draw conclusions about the current state of the network. These metrics also

provide information about network congestion and can be used to automatically tune the designed Net-VE. The tuning process includes send-rate change and packet resolution switching to enhance the performance. The automatic tuning is out of the scope of this thesis and is not implemented. The implementation purpose of the *NetworkAnalyzer* is network monitoring from an application-layer perspective.

The start of the application is very similar to *Pdu Server* and *Pdu Capture* program which is done by a batch (DOS) or bash (Linux) file using an initialization file written in XML. The sample initialization file is shown in Figure 5.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with XML Spy v4.3 by Ekrem Serin (NPS) -->
<System xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <DestinationAddress>129.174.65.60</DestinationAddress>
    <DestinationPort>9080</DestinationPort>
    <ReceivePort>9080</ReceivePort>
    <SendRateKbps>10</SendRateKbps>
    <PayloadInBytes>500</PayloadInBytes>
    <NumberOfPackets>2000</NumberOfPackets>
    <OutputFileURL>file://c:/xfsp/AnalyzeResults.txt</OutputFileURL>
</System>
```

Figure 5.7: *NetworkAnalyzer* Initialization File (Sender)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with XML Spy v4.3 by Ekrem Serin (NPS) -->
<System xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <DestinationAddress>131.120.7.1</DestinationAddress>
    <DestinationPort>9080</DestinationPort>
    <ReceivePort>9080</ReceivePort>
</System>
```

Figure 5.8: *NetworkAnalyzer* Initialization File (Receiver)

In order to collect previously mentioned metrics, a *Listener* must be up and running at the other end. *Listener* receives the packets and replays them back to the sender. A sample initialization file for listener is shown in Figure4.8.

The sender keeps track of each packet that it sends. When the packet is issued by the sender, current time for that packet is stored; at the time that sender gets the replayed

packet back, it updates the receive time for the packet. Packets that are not received back by the sender are considered as dropped.

At the end of the process, packets sent and received are analyzed to calculate the latency, drop-rate and jitter. The latency is calculated by dividing RTT (Round Trip Time) by two, the drop-rate is determined by keeping a counter for the packets that are not received and jitter is obtained by measuring variations between delays.

The fields of the initialization file in Figure 5.7 are self-descriptive. The send-rate field is slightly important from the network programmer perspective. The send-rate depends on the CPU cycle that the application runs on and the buffer allocation by the OS (Operating System) to the current process. In order to set the send-rate the sender is forced to sleep for the calculated sleep time which is measured by sending test packets to the local host. In order to test the accuracy of the designed program tests are conducted on a computer with P4 1.6GHz processor and 512 MB RAM (Random Access Memory). The conducted tests showed that send-rates up-to 400Kbps are accurately achieved by the designed program, and rates higher than that rate cannot be achieved due to OS system calls for sleep threads.

In network analyzing, there are three major points that the network analysis must consider. First, the current architecture of the Internet is a packet-switched network where packets can be routed via different routes resulting in out of order reception. Second, drops may occur at different places such as during transit from sender to receiver, at the receiver due to high send-rate and buffer limitation and during the transit from receiver to the sender. Third is that time-measurement in the Java programming language cannot be accurate for the values less than 10 milliseconds. The Java program cannot necessarily distinguish such precise differences due to the accuracy of various operating system calls.

F. SUMMARY

This chapter provides information about the implementation details of *Pdu Server*, *Pdu Capture* and *NetworkAnalyzer* programs. The implemented programs are described in detail to show the design of a simple PDU Farm. The PDU Farm is able to send and receive unicast as well as multicast messages. With this framework previously recorded scenarios can be replayed and analyzed. Another possible use of this work is simulating off-site entities without the need of rendering.

VI. EXPERIMENTS, DATA COLLECTION AND ANALYSIS

A. INTRODUCTION

This chapter examines the results and collected metrics from the conducted experiments.

B. OVERVIEW

In order to test the speed of XML Serialization program and collect the network metrics described previously, two sets of experiments were conducted. In the first set, the speed of XML Serialization program is tested by continuously generating serialized XML documents and sending them to the local host. In this set, the number of serialized documents that are sent and the number received are tracked. Additionally, the duration for XML Serialization is measured.

In the second set of experiments, latency, jitter and drop-rate were measured between George Mason University (GMU) (Fairfax, Virginia) and Monterey, California by using commercial Internet Service Providers (ISPs) on V.98 modem as well as on Ethernet. The Naval Postgraduate School (NPS) network infrastructure is not used for the initial set of experiments due to firewall problems described in the following sections.

C. XML SERIALIZATION PROGRAM

To test the speed of XML Serialization program, XML Serializer is programmed to continuously generate serialized XML documents. These binary XML documents (serialized) are sent to the local host listening on a specified UDP port.

The experiments are conducted on a machine whose properties are listed below. In the following list, the protocol and the buffer sizes are also specified.

Brand : Dell Inspiron 8200 Laptop

CPU and Memory : P4 1.6 GHz, 512 MByte

Transport Protocol : UDP

Send Buffer Size : 8192 Bytes

Receive Buffer Size : 8192 Bytes

XFSP Entity State PDU Size : 670 Bytes

This experiment shows the rate of binary XML document generation and the drop rate on UDP socket with 8192 Bytes receive buffer size. In this test the documents are generated continuously and are not regulated for send rate resulting in burst sending. The metrics collected for this set of experiment are shown in Table 6.1. Table 6.1 specifies the number of serialized XML documents that are sent and received as well as the total duration for serializing them.

Sent	Received	Duration (secs)	Binary XML Documents / Sec
10	10	0.2	50
100	100	0.47	212
200	200	0.89	224
300	300	1.18	254
400	400	1.47	272
500	425	1.73	289
600	524	1.97	304
700	468	2.12	330
800	409	2.39	334
900	474	2.67	337
1000	558	2.95	338
5000	1987	12.83	389
10000	3991	25.14	397
20000	7775	48.27	414

Table 6.1: XML Serialization Program Experiment

As seen from Table 6.1, the Serializer program reaches its maximum limit at 20000 serialized XML document row and starts to generate the XML documents at a rate ~410 documents / sec. The serialized XML documents were XFSP Entity State PDUs that occupied 670 Bytes in memory. The behavior of reaching its maximum limit is shown in Figure 6.1. Figure 6.1 reveals that XML Serializer program cannot generate

XFSP Entity State PDUs faster than ~410 documents /sec which is sufficient for fairly sophisticated Net-VEs.

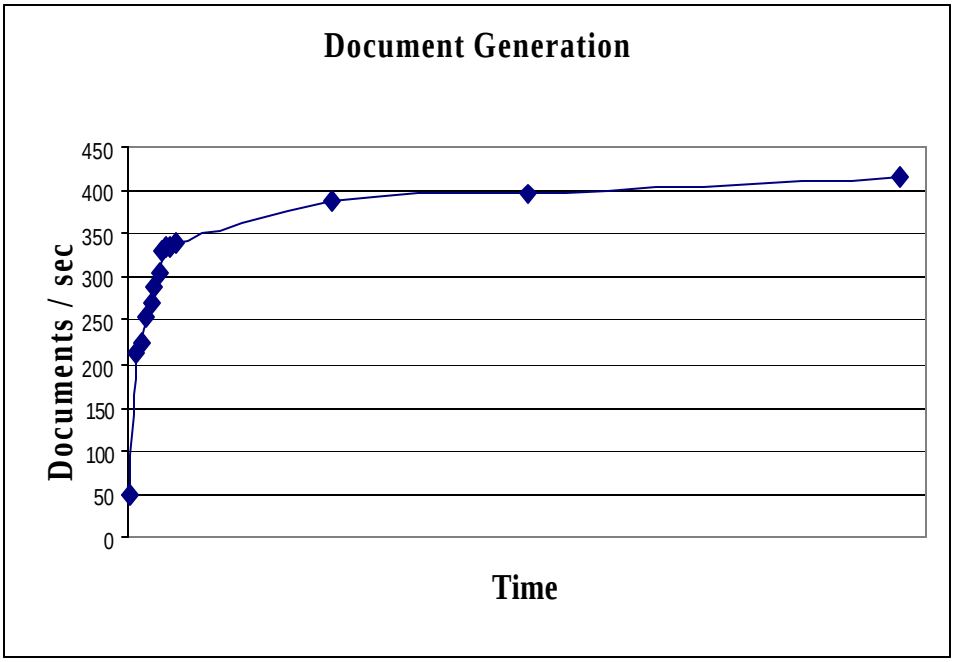

Figure 6.1: Maximum Limit of Binary XML Generation

Figure 6.2 shows the analysis of drop-rate for local host transmission. The drops in local host transmission occur due to buffer and CPU cycle limitations. In socket operations, an Operating System (OS) assigns a buffer for sending and receiving processes to each socket opened. For example, the default buffer value that Windows uses for UDP sockets is 8192 Bytes. When the program receives at a rate higher that it can handle, the received packets will be dropped until enough space is available in the buffer. As seen from Figure 6.2, the drop-rate on UDP socket with 8192 Bytes receive buffer is increasing as the send-rate increases.

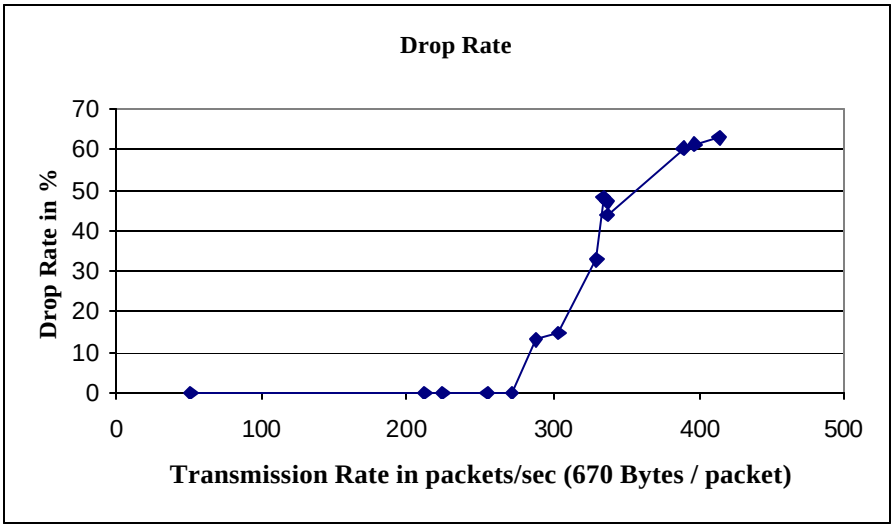

Figure 6.2: Drop Rates for Local Host Transmission

Figure 6.2 reveals that after reaching certain send-rate some packets will be ignored by the receiver.

D. NETWORK METRICS

In order to design rich and well-managed Net-VEs, application implementers need to consider basic metrics such as latency, drop-rate and jitter over different network topologies. These metrics help designers to manage their transmission and receive rates as well as the performance of the simulation that they implement. The effects of latency, drop-rate and jitter on a Net-VE are described previously. A Net-VE with poor network management will distract users and reduce their sense of immersion.

Two sets of experiments are conducted to measure the network metrics. For both experiments, the receive point was George Mason University (Fairfax, Virginia). In the first set of experiments, a commercial ISP is used by connecting via PPP (Point-to-Point Protocol) over V.98 voice modem. In the second set of experiments another commercial ISP is used by connecting via Ethernet over T1. The results achieved from the conducted experiments are presented and analyzed in the following sections.

1. V.98 Voice Modem

To test the V.98 voice modem transmission, two sets of experiments are conducted. At each experiment the send rate is changed and previously described metrics are collected. In the first experiment 2000 packets with 500 Bytes payload are sent at a rate of 10 Kbps. The collected metrics for this experiment are shown in Table 6.2.

Send Rate	Total Packets Sent	Drop Rate (%)	Average Latency (msecs)	Average Jitter (msecs)
10 Kbps	2000	2.0	146.6	2.54

Table 6.2: Metrics for 10Kbps Transmission on V.98 Modem

The measured latencies and jitter for this set of experiment are represented in Figure 6.3. and Figure 6.4 respectively.

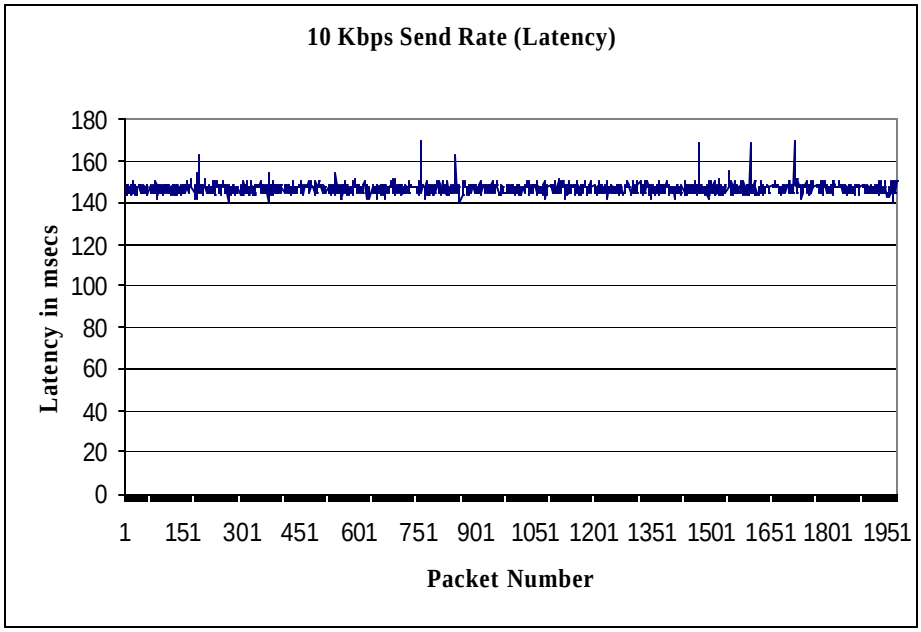

Figure 6.3: Cross-USA Latency in 10Kbps Send Rate on V.98 Voice Modem

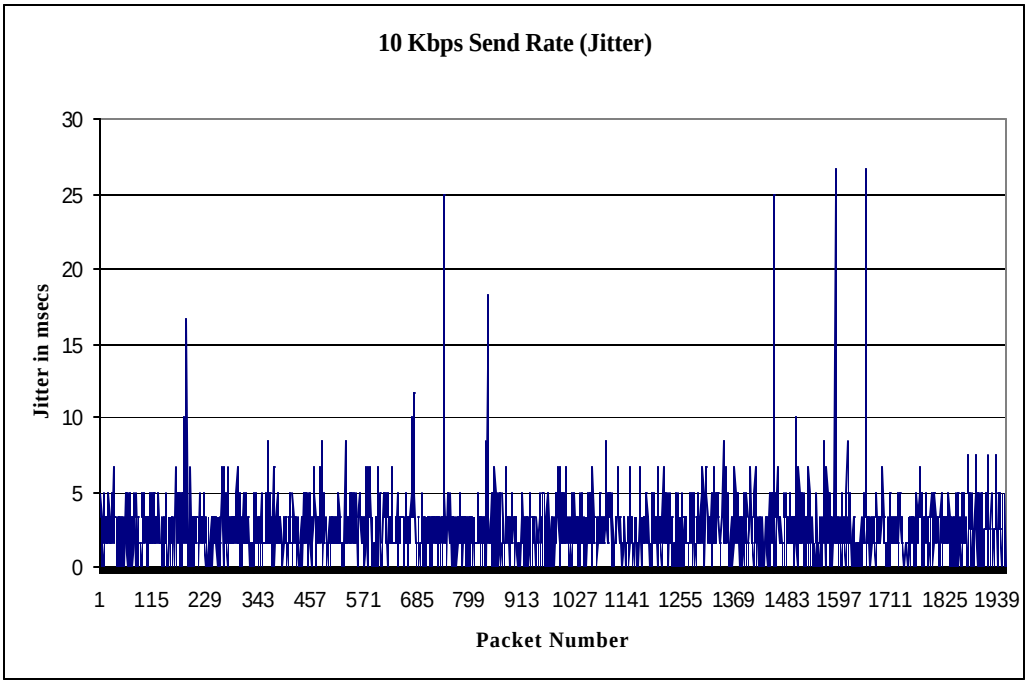

Figure 6.4: Cross-USA Jitter in 10Kbps Send Rate on V.98 Voice Modem

In the second set of experiments, 4000 packets with payload 500 bytes are sent at 50 Kbps rate. Drop-rate is measured as 1.18%. The average latency and jitter values are shown in Table 6.4.

Send Rate	Total Packets Sent	Drop Rate (%)	Average Latency (msecs)	Average Jitter (msecs)
50 Kbps	4000	1.18	158.32	15.62

Table 6.3 : Metrics for 50Kbps Transmission on V.98 Modem

The measured latencies and pure jitters for this set of experiment are represented in Figure 6.5. and Figure 6.6, respectively.

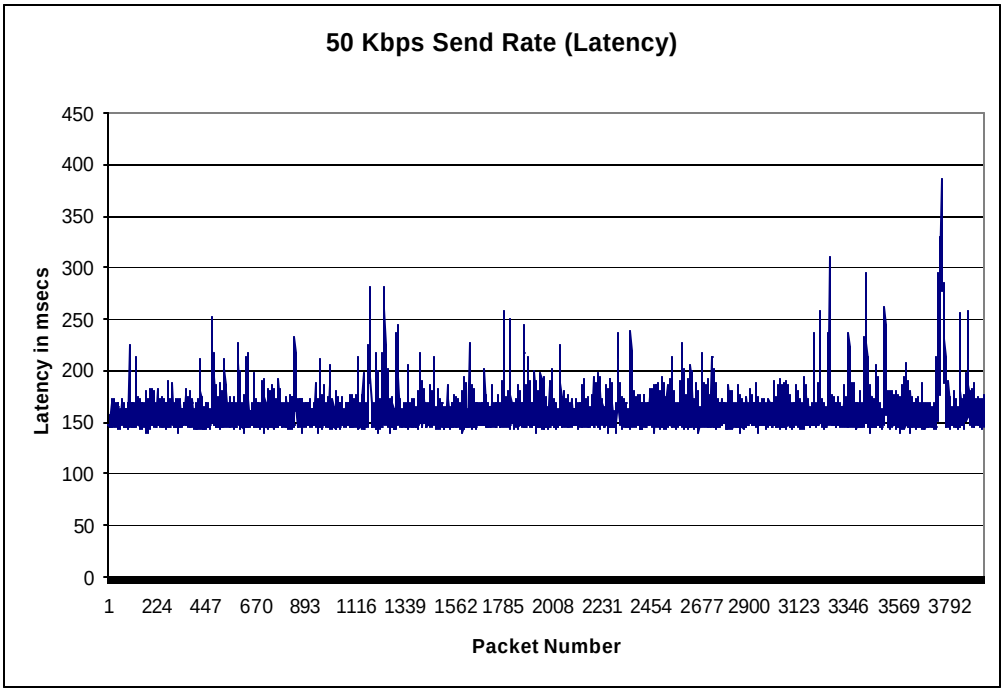

Figure 6.5: Cross-USA Latency in 50Kbps Send Rate on V.98 Voice Modem

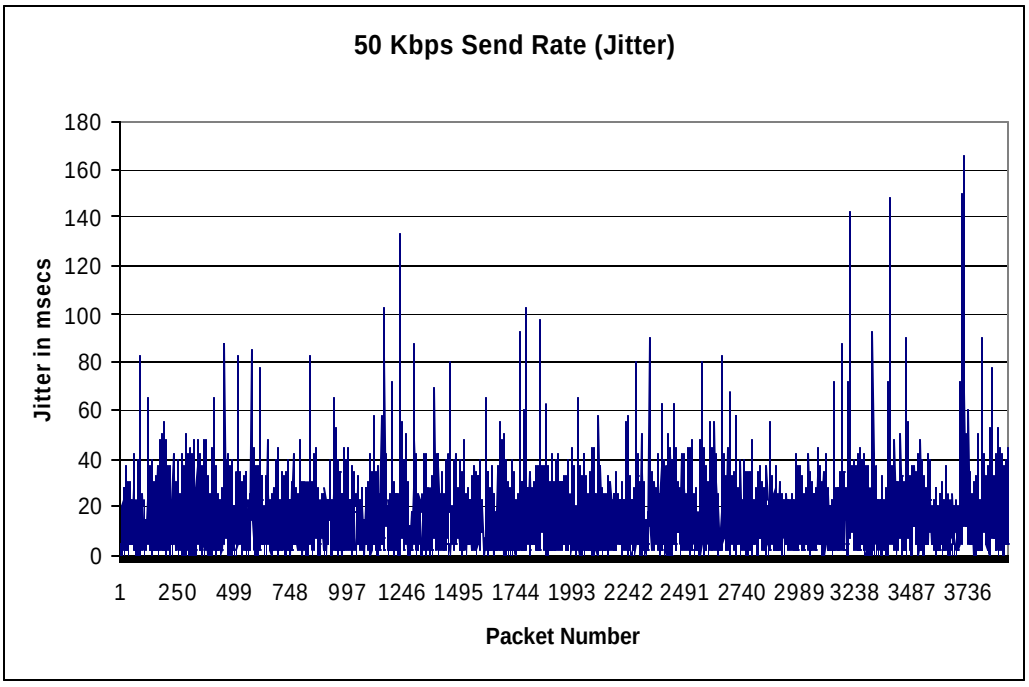

Figure 6.6: Cross-USA Jitter in 50Kbps Send Rate on V.98 Voice Modem

The experiments conducted on V.98 voice modem using PPP ISP connection revealed that average delay between Monterey, California and Fairfax, Virginia is over 140 milliseconds for a voice modem transmission. It also shows that the delay is not a constant process and can change due to network conditions. The Net-VE designer can use these metrics while designing their Net-VE to control the transmission rate.

2. Wide-Area Network (T1)

To test the Ethernet transmission, three sets of experiments are conducted. At each experiment, the send rate is changed and previously described metrics are collected. In the first experiment 2000 packets with 500 bytes payload are transmitted at a 10 Kbps send-rate. In this set, the drop-rate is measured as 1.13%. The collected metrics for this experiment are represented in Table 6.4.

Send Rate	Total Packets Sent	Drop Rate (%)	Average Latency (msecs)	Average Jitter (msecs)
10 Kbps	2000	1.13	51.9	2.84

Table 6.4: Metrics for 10Kbps Transmission on T1

The measured latencies and jitter for this set of experiment are represented in Figure 6.7. and Figure 6.8, respectively.

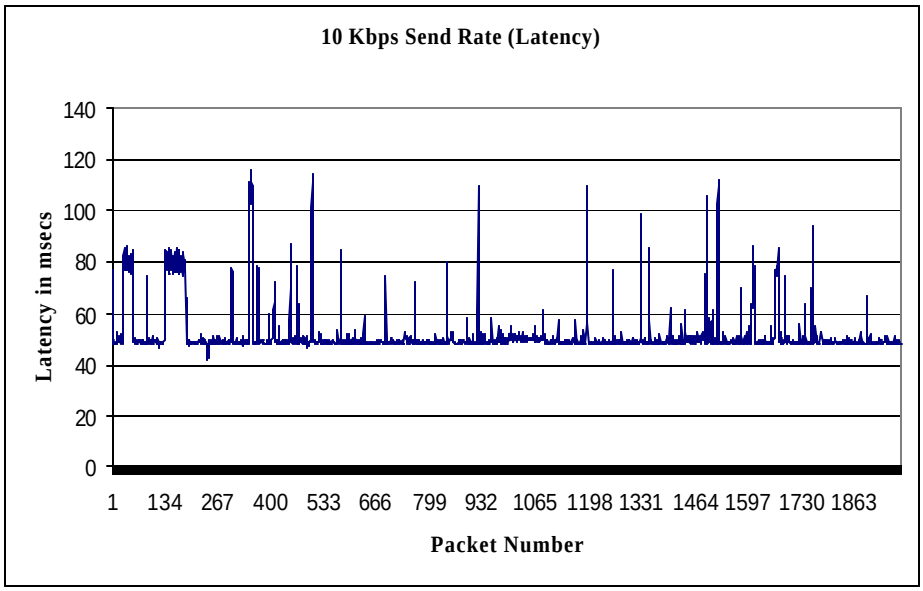

Figure 6.7: Latency in 10Kbps Send Rate on T1

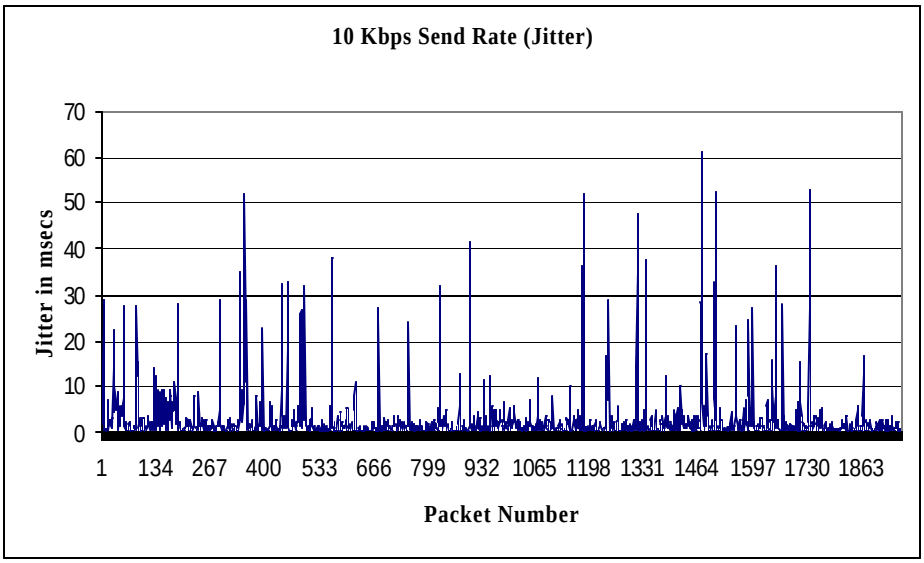

Figure 6.8: Jitter in 10Kbps Send Rate on T1

In the second set, 5000 packets with payload 500 bytes are sent at 100 Kbps rate. In this experiment the drop-rate is measured as 3.8%. The average latency and jitter values are shown in Table 6.5.

Send Rate	Total Packets Sent	Drop Rate (%)	Average Latency (msecs)	Average Jitter (msecs)
100 Kbps	5000	3.8	64.4	2.5

Table 6.5: Metrics for 100Kbps Transmission on Ethernet

The measured latencies and jitter for this set of experiment are represented in Figure 6.9. and Figure 6.10, respectively.

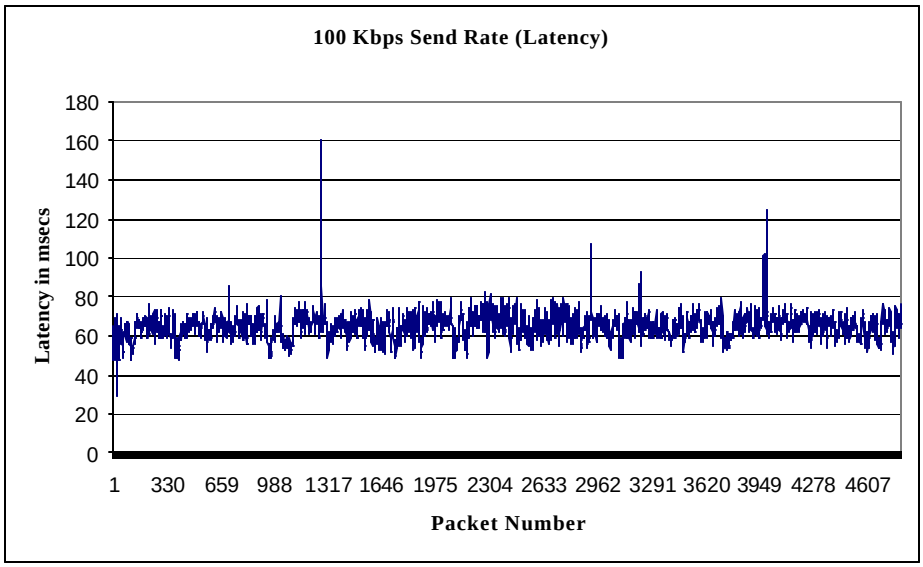

Figure 6.9: Latency in 100Kbps Send Rate on T1

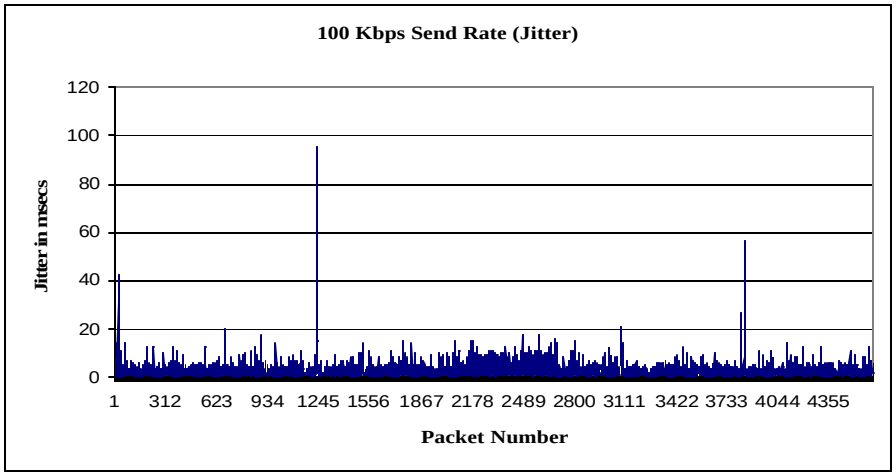

Figure 6.10: Jitter in 100Kbps Send Rate on T1

In the last set, 10000 packets with payload 500 bytes are sent at 400 Kbps rate. In this experiment the drop-rate is measured as 4.76%. The average latency and jitter values are shown in Table 6.6.

Send Rate	Total Packets Sent	Drop Rate (%)	Average Latency (msecs)	Average Jitter (msecs)
400 Kbps	10000	4.76	217.8	0.4

Table 6.6: Metrics for 400Kbps Transmission on Ethernet

The measured latencies and jitter for this set of experiment are represented in Figure 6.11. and Figure 6.12 respectively. The latency began at values similar to earlier runs, but increased over time. The most probable cause of this is increasing queue size in routers. At low data rates the queues stay relatively empty. As the data rate crosses the queue service threshold the queue grows to its capacity and starts dropping the received packets. There are various schemes for handling long queue sizes (e.g. dropping packets) in routers. Probably the most popular is *tail-drop*. In this scheme, the packet queue size grows until there is no room in the queue, and the arriving packets are simply dropped. Another popular dropping scheme is Random Early Drop (RED). In that scheme, when the queue size reaches some size smaller than its full capacity, the packets to be dropped are selected randomly. The probability of drop for both schemes increases as the queue size increases to its capacity.

Another logical explanation for exponential increase in latency is packets to be delivered using different routes. When the link between two hops becomes congested, the routers can choose the packets to follow different hops resulting in different latency values for different packets. The calculation of using different route will also put some latency over the packets during their journey from source to destination.

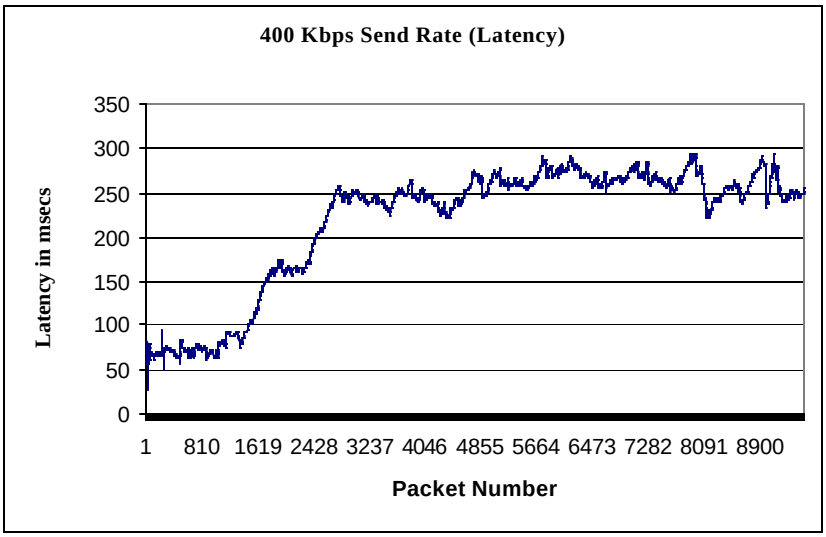

Figure 6.11: Latency in 400Kbps Send Rate on T1 (Day-1)

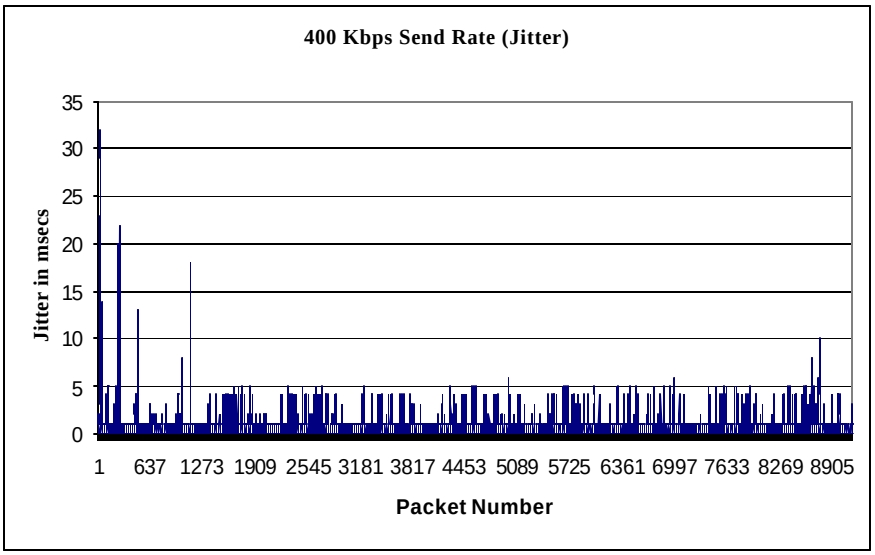

Figure 6.12: Jitter in 400Kbps Send Rate on T1

The experiments conducted to analyze the exponential increase behavior gave similar results one on different days, Figure 6.13 and Figure 6.14. The routes from George Mason University to `MovesInstitute.org` are also presented to track the problem. This type of behavior can be caused by any of the nodes listed in Figure 6.15. The *traceroute* packets followed the same path presented in Figure 6.15 before and after

high rate transmission. This data reveals that the probable cause of latency problem is drops occurring in routers.

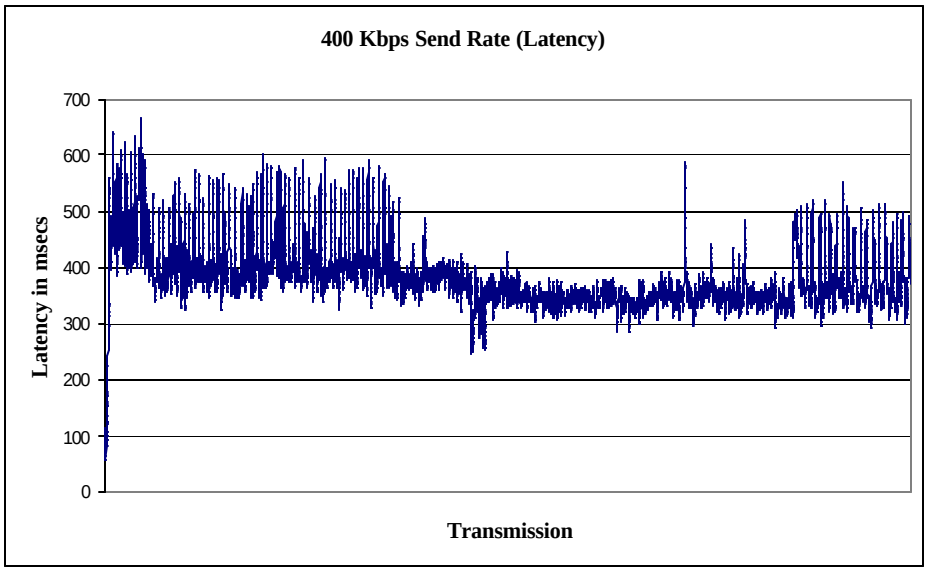

Figure 6.13: Latency in 400Kbps Send Rate on T1 (Day -2)

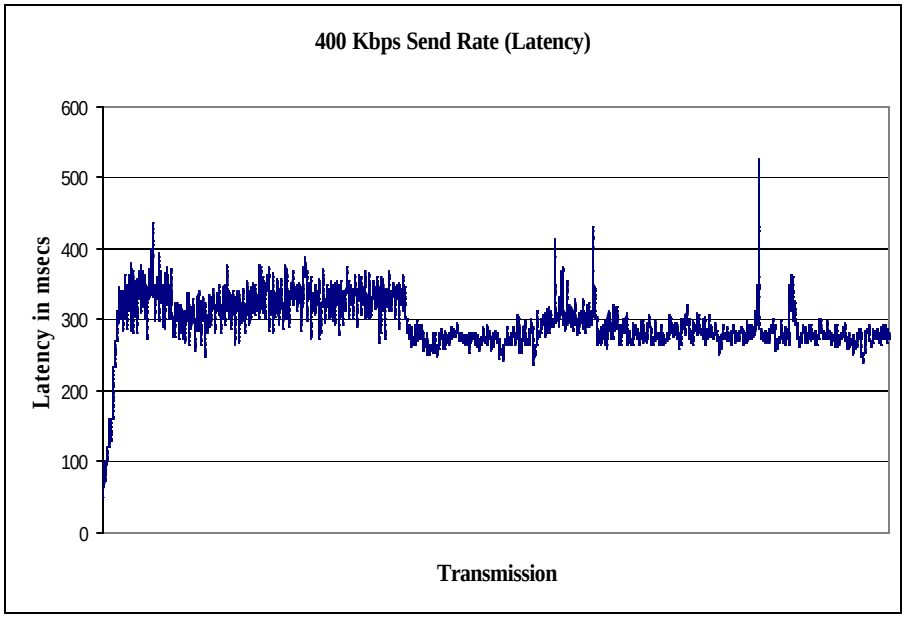

Figure 6.14: Latency in 400Kbps Send Rate on T1 (Day -3)

```
[netlab] # traceroute movesinstitute.org

 1  netlab (129.174.65.1)  1.174 ms  0.793 ms  0.741 ms
 2  129.174.249.129 (129.174.249.129)  0.871 ms  0.735 ms  0.694 ms
 3  129.174.248.233 (129.174.248.233)  1.115 ms  1.176 ms  1.109 ms
 4  129.174.247.118 (129.174.247.118)  0.855 ms  0.605 ms  0.861 ms
 5  WTN1-GeorgeMasonUFairfax.networkvirginia.net (65.162.90.5)  6.902 ms  6.662
ms  5.833 ms
 6  65.162.89.38 (65.162.89.38)  10.842 ms  10.816 ms  10.539 ms
 7  sl-gw20-rly-2-2.sprintlink.net (160.81.255.1)  11.820 ms  11.909 ms  11.072
ms
 8  sl-bb23-rly-3-2.sprintlink.net (144.232.14.45)  142.828 ms  205.470 ms  244.
164 ms
 9  sl-bb21-pen-12-0.sprintlink.net (144.232.20.33)  21.815 ms  17.643 ms  13.12
1 ms
10  sl-bb20-pen-15-0.sprintlink.net (144.232.16.33)  17.500 ms  17.448 ms  13.96
7 ms
11  sl-bb20-stk-10-0.sprintlink.net (144.232.18.46)  79.529 ms  76.489 ms *
12  sl-gw25-stk-9-0.sprintlink.net (144.232.4.218)  73.191 ms  71.910 ms  73.974
 ms
13  sl-swb-57-0.sprintlink.net (144.223.59.86)  80.435 ms  81.463 ms  78.775 ms
14  ded1-fa5-1-0.mtry01.pbi.net (206.171.158.133)  83.213 ms  81.779 ms  77.889
ms
15  Naval-Post-Graduate-School-505769.cust-rtr.pacbell.net (209.232.139.54)  86.
757 ms  81.758 ms  80.937 ms
16  63.205.26.77 (63.205.26.77)  88.010 ms  79.443 ms  82.471 ms
```

Figure 6.15: Traceroute From `gmu.edu` To `MovesInstitute.org`

E. NPS FIREWALL PROBLEM

Experiments using Naval Postgraduate School network backbone were infrequently conducted due to firewall problems. To find the reasons of firewall problem, the *traceroute* program is used. This program shows all the nodes that *traceroute* packets hit during their journey from source to destination.

When the computer behind Naval Postgraduate School firewall was "*tracerouted*", firewall acted as a proxy and sent Internet Control Message Protocol (ICMP) packets to the *traceroute* initiator. In this case, the traceroute initiator assumed that the packets could be delivered to the computer behind the firewall. Unfortunately, when the experiments were attempted, the firewall intercepted every packet destined for the computer behind it, and did not forward them. Although the first proxy behavior is

consistent with firewalls that try to hide its internal topology, the second behavior is not consistent and should be resolved. Firewall adjustments and further experiments need to continue.

F. SUMMARY

This chapter summarizes the data collected in conducted experiments. For data collection two main sets of experiments are conducted. In the first set the speed of XML Serializer program is measured, and in the second, network metrics are measured by using V.98 voice modem and Ethernet transmission. Furthermore, the results achieved are represented in 2D graphs.

This chapter also presents the usage of Network Analyzer program. This program can be used as proxy to track the changes in the latency where the changes can serve as basis to find the maximum capacity of the link. Additionally, the capacity of the link can be used to better manage the network transmissions resulting in richer Net-VEs.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND FUTURE WORK

A. CONCLUSION

1. Application Layer Protocol Extensibility

Application layer protocols can be extended at run-time by using a protocol definition language. The run-time extensibility of the application layer protocols provides a way to optimize the bandwidth usage and to meet the needs of different users at different fidelity levels.

Extensibility in Net-VE architectures is essential because it is clear that networks will continue to evolve, and a closed system which cannot be changed would be obsolete as the networking technology evolves.

In this framework, XML Schema is used and considered as a good candidate to be protocol definition language. XML Schema can well describe the application protocol syntax with its internal as well as user-defined data structures. In order to parse the protocol definition document written in XML Schema, a schema parser is implemented.

2. XML Serialization and XML Deserialization

XML Serialization provides a compact way to send and receive XML documents over the network. Currently most of XML documents use UTF-8 encoding which corresponds to the 8-bit ASCII encoding. With this scheme each alphabetical character and number is represented by eight bits. Instead of using 8 bits for each character, the notion *"agreement"* is exploited and element and attribute names are replaced by binary short tags. Furthermore, the data marked-up by elements and attributes are serialized to binary form resulting in compact XML.

With the implemented algorithm XML documents are compressed without using any binary compression algorithms.

Driving factors of avoiding binary compression algorithms are simplicity of implementation, computational speed performance and skipping bitwise operations by

providing reimplementability and generality. Further elaborations can complicate the protocol beyond the comprehensibility except for information encoding experts.

The compressed XML document is called as binary XML and decompressed by the XML Deserializer. The result of the decompression is the text XML document provided to the XML Serializer.

With XML Serializer and XML Deserializer XML documents are sent in a more compact way over the network providing bandwidth and time saving.

3. PDU Farm and Network Monitoring

In this thesis a simple PDU Farm and Network Monitoring is implemented. PDU Farms let users to test the designed Net-VE for 24 hours a day and 7 days a week. Furthermore, they provide a way to mimic the previously played scenarios to draw tactical as well as technical conclusions.

The technical conclusions derived from the played Net-VE simulation include the metrics for the bandwidth consumption, rendering performance and memory usage. The recorded scenario can also be replayed multiple times to accurately measure these metrics resulting in enhancing the design of Net-VE.

The network monitoring program measures the current state of the network and provides information about network congestion, latency, drop-rate and jitter. These metrics are highly used by network programmers and can provide a fundamental for tuning purposes mentioned in the recommendations for future work section.

B. RECOMMENDATIONS FOR FUTURE WORK

1. General XML Serializer / Deserializer

XML Serialization and Deserialization can be enhanced to compress and decompress any XML document defined by any XML Schema. Currently, XFSP is an ongoing project and will be enhanced to base the generation of binary XML documents. The recommended process for this enhancement includes;

- A schema validator implementation where it validates the provided schema.
- Namespace handling

This work needs to be continued as a general purpose XML compressor for both network and file streams.

2. Monitoring Agents

An *agent* is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objectives [Wooldridge 01]. To automatically tune the networking and change the application layer protocol in a Net-VE, the software agents can be used.

The network monitoring process can be implemented as an autonomous system and incorporated into the designed Net-VE system. These autonomous agents can work together and create a complex adaptive system, where that system can change the application layer protocol by using environmental information such as delay, drop-rate and jitter.

The complex adaptive system can optimize the network utilization and provide a better managed network on which rich and responsive large-scale networked virtual environments can be implemented.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. DIS SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with XML Spy v4.3 by Ekrem Serin (NPS) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="dis">
        <xs:complexType>
            <xs:choice>
                <xs:element name="espdu" type="espduType"/>
                <xs:element name="collisionpdu" type="collisionpduType"/>
                <xs:element name="firepdu" type="firepduType"/>
                <xs:element name="detonationpdu" type="detonationpduType"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="espduType">
        <xs:sequence>
            <xs:element name="header" type="headerType"/>
            <xs:element name="entityID" type="entityIDType"/>
            <xs:element name="forceID" type="xs:byte"/>
            <xs:element name="articulationNumber" type="xs:byte"/>
            <xs:element name="entityType" type="entityTypeType"/>
            <xs:element name="alternativeEntityType" type="entityTypeType"/>
            <xs:element name="linearVelocity" type="Vector3Float"/>
            <xs:element name="location" type="locationType"/>
            <xs:element name="orientation" type="orientationType"/>
            <xs:element name="appearence" type="xs:int"/>
            <xs:element name="deadReckoning" type="deadReckoningType"/>
            <xs:element name="entityMarking" type="entityMarkingType"/>
            <xs:element name="capabilities" type="capabilitiesType"/>
            <xs:element name="articulationParameters" type="articulationParameterType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="collisionpduType">
        <xs:sequence>
            <xs:element name="header" type="headerType"/>
            <xs:element name="issuingEntityID" type="entityIDType"/>
            <xs:element name="collidingEntityID" type="entityIDType"/>
            <xs:element name="eventID" type="eventIDType"/>
            <xs:element name="collisionType" type="xs:byte"/>
            <xs:element name="padding" type="xs:byte"/>
            <xs:element name="velocity" type="Vector3Float"/>
            <xs:element name="mass" type="xs:float"/>
            <xs:element name="location" type="Vector3Float"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="firepduType">
```

```
    <xs:sequence>
        <xs:element name="header" type="headerType"/>
        <xs:element name="firingEntityID" type="entityIDType"/>
        <xs:element name="targetEntityID" type="entityIDType"/>
        <xs:element name="munitionID" type="entityIDType"/>
        <xs:element name="eventID" type="eventIDType"/>
        <xs:element name="fireMissionIndex" type="xs:int"/>
        <xs:element name="locationInWorld" type="locationType"/>
        <xs:element name="burstDescriptor" type="burstDescriptorType"/>
        <xs:element name="velocity" type="Vector3Float"/>
        <xs:element name="range" type="xs:float"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="detonationpduType">
    <xs:sequence>
        <xs:element name="header" type="headerType"/>
        <xs:element name="firingEntityID" type="entityIDType"/>
        <xs:element name="targetEntityID" type="entityIDType"/>
        <xs:element name="munitionID" type="entityIDType"/>
        <xs:element name="eventID" type="eventIDType"/>
        <xs:element name="velocity" type="Vector3Float"/>
        <xs:element name="locationInWorld" type="locationType"/>
        <xs:element name="burstDescriptor" type="burstDescriptorType"/>
        <xs:element name="locationInEntity" type="Vector3Float"/>
        <xs:element name="detonationResult" type="xs:byte"/>
        <xs:element name="articulationNumber" type="xs:byte"/>
        <xs:element name="padding" type="xs:short"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="headerType">
    <xs:sequence>
        <xs:element name="protocolVersion" type="xs:byte"/>
        <xs:element name="exerciseID" type="xs:byte"/>
        <xs:element name="pduType" type="xs:byte"/>
        <xs:element name="protocolFamily" type="xs:byte"/>
        <xs:element name="timeStamp" type="xs:int"/>
        <xs:element name="length" type="xs:short"/>
        <xs:element name="padding" type="xs:short"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="entityIDType">
    <xs:sequence>
        <xs:element name="site" type="xs:short"/>
        <xs:element name="application" type="xs:short"/>
        <xs:element name="entity" type="xs:short"/>
    </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="entityTypeType">
    <xs:sequence>
        <xs:element name="kind" type="xs:byte"/>
        <xs:element name="domain" type="xs:byte"/>
        <xs:element name="country" type="xs:short"/>
        <xs:element name="category" type="xs:byte"/>
        <xs:element name="subcategory" type="xs:byte"/>
        <xs:element name="specific" type="xs:byte"/>
        <xs:element name="extra" type="xs:byte"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="locationType">
    <xs:attribute name="x" type="xs:double"/>
    <xs:attribute name="y" type="xs:double"/>
    <xs:attribute name="z" type="xs:double"/>
</xs:complexType>
<xs:complexType name="Vector3Float">
    <xs:attribute name="x" type="xs:float"/>
    <xs:attribute name="y" type="xs:float"/>
    <xs:attribute name="z" type="xs:float"/>
</xs:complexType>
<xs:complexType name="orientationType">
    <xs:attribute name="psi" type="xs:float"/>
    <xs:attribute name="theta" type="xs:float"/>
    <xs:attribute name="phi" type="xs:float"/>
</xs:complexType>
<xs:complexType name="deadReckoningType">
    <xs:sequence>
        <xs:element name="algorithm" type="xs:byte"/>
        <xs:element name="otherParameters" type="otherParametersType"/>
        <xs:element name="linearAcceleration" type="Vector3Float"/>
        <xs:element name="linearAngularVelocity" type="Vector3Float"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="otherParametersType">
    <xs:attribute name="op1" type="xs:byte"/>
    <xs:attribute name="op2" type="xs:byte"/>
    <xs:attribute name="op3" type="xs:byte"/>
    <xs:attribute name="op4" type="xs:byte"/>
    <xs:attribute name="op5" type="xs:byte"/>
    <xs:attribute name="op6" type="xs:byte"/>
    <xs:attribute name="op7" type="xs:byte"/>
    <xs:attribute name="op8" type="xs:byte"/>
    <xs:attribute name="op9" type="xs:byte"/>
    <xs:attribute name="op10" type="xs:byte"/>
    <xs:attribute name="op11" type="xs:byte"/>
    <xs:attribute name="op12" type="xs:byte"/>
```

```
    <xs:attribute name="op13" type="xs:byte"/>
    <xs:attribute name="op14" type="xs:byte"/>
    <xs:attribute name="op15" type="xs:byte"/>
</xs:complexType>
<xs:complexType name="entityMarkingType">
    <xs:sequence>
        <xs:element name="characterSet" type="xs:byte"/>
        <xs:element name="unsignedInts" type="unsignedIntsType"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="unsignedIntsType">
    <xs:attribute name="op1" type="xs:byte"/>
    <xs:attribute name="op2" type="xs:byte"/>
    <xs:attribute name="op3" type="xs:byte"/>
    <xs:attribute name="op4" type="xs:byte"/>
    <xs:attribute name="op5" type="xs:byte"/>
    <xs:attribute name="op6" type="xs:byte"/>
    <xs:attribute name="op7" type="xs:byte"/>
    <xs:attribute name="op8" type="xs:byte"/>
    <xs:attribute name="op9" type="xs:byte"/>
    <xs:attribute name="op10" type="xs:byte"/>
    <xs:attribute name="op11" type="xs:byte"/>
</xs:complexType>
<xs:complexType name="capabilitiesType">
    <xs:attribute name="op1" type="xs:boolean"/>
    <xs:attribute name="op2" type="xs:boolean"/>
    <xs:attribute name="op3" type="xs:boolean"/>
    <xs:attribute name="op4" type="xs:boolean"/>
    <xs:attribute name="op5" type="xs:boolean"/>
    <xs:attribute name="op6" type="xs:boolean"/>
    <xs:attribute name="op7" type="xs:boolean"/>
    <xs:attribute name="op8" type="xs:boolean"/>
    <xs:attribute name="op9" type="xs:boolean"/>
    <xs:attribute name="op10" type="xs:boolean"/>
    <xs:attribute name="op11" type="xs:boolean"/>
    <xs:attribute name="op12" type="xs:boolean"/>
    <xs:attribute name="op13" type="xs:boolean"/>
    <xs:attribute name="op14" type="xs:boolean"/>
    <xs:attribute name="op15" type="xs:boolean"/>
    <xs:attribute name="op16" type="xs:boolean"/>
    <xs:attribute name="op17" type="xs:boolean"/>
    <xs:attribute name="op18" type="xs:boolean"/>
    <xs:attribute name="op19" type="xs:boolean"/>
    <xs:attribute name="op20" type="xs:boolean"/>
    <xs:attribute name="op21" type="xs:boolean"/>
    <xs:attribute name="op22" type="xs:boolean"/>
    <xs:attribute name="op23" type="xs:boolean"/>
```

```
            <xs:attribute name="op24" type="xs:boolean"/>
            <xs:attribute name="op25" type="xs:boolean"/>
            <xs:attribute name="op26" type="xs:boolean"/>
            <xs:attribute name="op27" type="xs:boolean"/>
            <xs:attribute name="op28" type="xs:boolean"/>
            <xs:attribute name="op29" type="xs:boolean"/>
            <xs:attribute name="op30" type="xs:boolean"/>
            <xs:attribute name="op31" type="xs:boolean"/>
            <xs:attribute name="op32" type="xs:boolean"/>
        </xs:complexType>
        <xs:complexType name="eventIDType">
            <xs:sequence>
                <xs:element name="site" type="xs:short"/>
                <xs:element name="application" type="xs:short"/>
                <xs:element name="eventNumber" type="xs:short"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="burstDescriptorType">
            <xs:sequence>
                <xs:element name="munition" type="xs:long"/>
                <xs:element name="warhead" type="xs:short"/>
                <xs:element name="fuze" type="xs:short"/>
                <xs:element name="quantity" type="xs:short"/>
                <xs:element name="rate" type="xs:short"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="articulationParameterType">
            <xs:sequence>
                <xs:element name="typeDesignator" type="xs:byte"/>
                <xs:element name="changeIndicator" type="xs:byte"/>
                <xs:element name="id" type="xs:short"/>
                <xs:element name="parameterType" type="xs:integer"/>
                <xs:element name="parameterValue" type="xs:long"/>
            </xs:sequence>
        </xs:complexType>
    </xs:schema>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. ENTITY STATE PDU EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<espdu xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\XFSP\source\org\npsnet\xfsp\schemas\espdu.xsd
">
    <header>
        <protocolVersion>1</protocolVersion>
        <exerciseID>2</exerciseID>
        <pduType>3</pduType>
        <protocolFamily>4</protocolFamily>
        <timeStamp>1000</timeStamp>
        <length>400</length>
        <padding>300</padding>
    </header>
    <entityID>
        <site>50</site>
        <application>60</application>
        <entity>70</entity>
    </entityID>
    <forceID>19</forceID>
    <articulationNumber>10</articulationNumber>
    <entityType>
        <kind>1</kind>
        <domain>1</domain>
        <country>99</country>
        <category>10</category>
        <subcategory>3</subcategory>
        <specific>5</specific>
        <extra>7</extra>
    </entityType>
    <alternativeEntityType>
        <kind>1</kind>
        <domain>2</domain>
        <country>89</country>
        <category>2</category>
        <subcategory>5</subcategory>
        <specific>7</specific>
        <extra>5</extra>
    </alternativeEntityType>
    <linearVelocity x="12.90" y="1.6" z="0.0"/>
    <location x="233.56" y="5.7" z="1.8"/>
    <orientation phi="0.6" theta="0.5" psi="0.0"/>
    <appearence>12345</appearence>
    <deadReckoning>
        <algorithm>4</algorithm>
```

```
        <otherParameters op1="1" op10="10" op11="11" op12="12" op13="13"
op14="14" op15="15" op2="2" op3="3" op4="4" op5="5" op6="6" op7="7" op8="8"
op9="9"/>
        <linearAcceleration x="1.0" y="1.0" z="1.0"/>
        <linearAngularVelocity x="2.0" y="2.0" z="2.0"/>
    </deadReckoning>
    <entityMarking>
        <characterSet>23</characterSet>
        <unsignedInts op1="1" op2="2" op3="3" op11="11" op10="10" op4="4" op5="5"
op6="6" op7="7" op8="8" op9="9"/>
    </entityMarking>
    <capabilities op1="true" op10="true" op11="false" op12="false" op13="true"
op14="false" op15="false" op16="false" op17="false" op18="false" op19="false"
op2="false" op20="false" op21="false" op22="false" op23="true" op24="false"
op25="false" op26="false" op27="false" op28="false" op29="false" op3="false"
op30="false" op31="false" op32="false" op4="true" op5="true" op6="true" op7="false"
op8="false" op9="false"/>
    <articulationParameters>
        <typeDesignator>1</typeDesignator>
        <changeIndicator>2</changeIndicator>
        <id>5</id>
        <parameterType>10</parameterType>
        <parameterValue>7</parameterValue>
    </articulationParameters>
</espdu>
```

APPENDIX C. DETONATION PDU EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<detonationpdu xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\XFSP\source\org\npsnet\xfsp\schemas\detonation
pdu.xsd">
    <header>
        <protocolVersion>9</protocolVersion>
        <exerciseID>9</exerciseID>
        <pduType>9</pduType>
        <protocolFamily>9</protocolFamily>
        <timeStamp>12345678</timeStamp>
        <length>234</length>
        <padding>33</padding>
    </header>
    <firingEntityID>
        <site>1</site>
        <application>1</application>
        <entity>99</entity>
    </firingEntityID>
    <targetEntityID>
        <site>1</site>
        <application>1</application>
        <entity>89</entity>
    </targetEntityID>
    <munitionID>
        <site>1</site>
        <application>2</application>
        <entity>3</entity>
    </munitionID>
    <eventID>
        <site>1</site>
        <application>1</application>
        <eventNumber>99</eventNumber>
    </eventID>
    <velocity x="10.3" y="44.2" z="67.7"/>
    <locationInWorld x="22.1" y="33.5" z="12.9"/>
    <burstDescriptor>
        <munition>2222222</munition>
        <warhead>111</warhead>
        <fuze>333</fuze>
        <quantity>444</quantity>
        <rate>555</rate>
    </burstDescriptor>
    <locationInEntity x="123.456" y="456.789" z="980.123"/>
    <detonationResult>10</detonationResult>
```

```
    <articulationNumber>0</articulationNumber>
    <padding>111</padding>
</detonationpdu>
```

APPENDIX D. FIRE PDU EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<firepdu xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\XFSP\source\org\npsnet\xfsp\schemas\firepdu.xsd">
    <header>
        <protocolVersion>1</protocolVersion>
        <exerciseID>2</exerciseID>
        <pduType>3</pduType>
        <protocolFamily>4</protocolFamily>
        <timeStamp>555</timeStamp>
        <length>6</length>
        <padding>77</padding>
    </header>
    <firingEntityID>
        <site>1</site>
        <application>1</application>
        <entity>10</entity>
    </firingEntityID>
    <targetEntityID>
        <site>1</site>
        <application>1</application>
        <entity>11</entity>
    </targetEntityID>
    <munitionID>
        <site>1</site>
        <application>1</application>
        <entity>10</entity>
    </munitionID>
    <eventID>
        <site>1</site>
        <application>1</application>
        <eventNumber>22</eventNumber>
    </eventID>
    <fireMissionIndex>123567</fireMissionIndex>
    <locationInWorld x="464646.899" y="4646.28" z="19997.567"/>
    <burstDescriptor>
        <munition>47747884</munition>
        <warhead>345</warhead>
        <fuze>125</fuze>
        <quantity>100</quantity>
        <rate>10</rate>
    </burstDescriptor>
    <velocity x="10" y="10" z="10"/>
    <range>19929.991</range>
</firepdu>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. COLLISION PDU EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<collisionpdu xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\XFSP\source\org\npsnet\xfsp\schemas\collisionp
du.xsd">
    <header>
        <protocolVersion>1</protocolVersion>
        <exerciseID>2</exerciseID>
        <pduType>3</pduType>
        <protocolFamily>4</protocolFamily>
        <timeStamp>5555</timeStamp>
        <length>34</length>
        <padding>444</padding>
    </header>
    <issuingEntityID>
        <site>10</site>
        <application>20</application>
        <entity>30</entity>
    </issuingEntityID>
    <collidingEntityID>
        <site>40</site>
        <application>50</application>
        <entity>60</entity>
    </collidingEntityID>
    <eventID>
        <site>1</site>
        <application>1</application>
        <eventNumber>101</eventNumber>
    </eventID>
    <collisionType>110</collisionType>
    <padding>33</padding>
    <velocity x="2.0" y="2.9" z="3.0"/>
    <mass>34.67</mass>
    <location x="101.01" y="44.67" z="12.10"/>
</collisionpdu>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. HIERARCHY OF ALL PACKAGES

Package Hierarchies:

org.npsnet.xfsp,
org.npsnet.xfsp.datatypes,
org.npsnet.xfsp.pduserver,
org.npsnet.xfsp.swing,
org.npsnet.xfsp.tests

Class Hierarchy:

- class org.npsnet.xfsp.**BinaryReader**
- class org.npsnet.xfsp.**BinaryReaderX3D**
- class org.npsnet.xfsp.datatypes.**ComplexType** (implements org.npsnet.xfsp.datatypes.Type)
- class org.npsnet.xfsp.pduserver.**Help**
- class org.npsnet.xfsp.swing.**LoaderDemo**
- class org.npsnet.xfsp.pduserver.**PDUServerWithGUI**
- class org.npsnet.xfsp.tests.**ReceiverSimulation**
- class org.npsnet.xfsp.tests.**SenderSimulation**
- class org.npsnet.xfsp.tests.**Compressor**
- class org.npsnet.xfsp.**DocumentProcessor**
- class org.npsnet.xfsp.**DocumentProcessorX3D**
- class org.npsnet.xfsp.**DOMManipulator**
- class org.npsnet.xfsp.**DOMManipulatorX3D**
- class javax.swing.filechooser.FileFilter
- class org.npsnet.xfsp.pduserver.**PDUServerWithGUI.PDUFileFilter**
- class org.npsnet.xfsp.pduserver.**PDUServerWithGUI.XSDFileFilter**
- class org.npsnet.xfsp.tests.**ReceiverSimulation.SchemaFileFilter**
- class org.npsnet.xfsp.tests.**SenderSimulation.XMLFileFilter**
- class org.npsnet.xfsp.datatypes.**MFBool** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFColor** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFDouble** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFFloat** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFImage** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFInt32** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFRotation** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFString** (implements org.npsnet.xfsp.datatypes.SimpleType)

- class org.npsnet.xfsp.datatypes.**MFTime** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFVec2d** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFVec2f** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFVec3d** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**MFVec3f** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.pduserver.**MyTimer** (implements java.lang.Runnable)
- class org.npsnet.xfsp.pduserver.**NetworkAnalyzerReceiver**
- class org.npsnet.xfsp.pduserver.**NetworkAnalyzerSender**
- class org.npsnet.xfsp.pduserver.**Packet**
- class org.npsnet.xfsp.pduserver.**PDUCapturer** (implements java.lang.Runnable)
- class org.npsnet.xfsp.pduserver.**PDUServer**
- class org.npsnet.xfsp.pduserver.**PDUServerWithGUI.PDUSender** (implements java.lang.Runnable)
- class org.npsnet.xfsp.datatypes.**SFBool** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFColor** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFDouble** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFFloat** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFImage** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFInt32** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFRotation** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFString** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFTime** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFVec2d** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFVec2f** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFVec3d** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**SFVec3f** (implements org.npsnet.xfsp.datatypes.SimpleType)

- class org.npsnet.xfsp.datatypes.**SimpleTypeExtension** (implements org.npsnet.xfsp.datatypes.SimpleTypeWithName)
- class org.npsnet.xfsp.**TableAttribute**
- class org.npsnet.xfsp.**TableAttributeX3D**
- class org.npsnet.xfsp.**TableElement**
- class org.npsnet.xfsp.**TableElementX3D**
- class org.npsnet.xfsp.**TableManager**
- class org.npsnet.xfsp.**TableManagerX3D**
- class org.npsnet.xfsp.datatypes.**TypeFactory**
- class org.npsnet.xfsp.swing.**XMLSwingTree**
- class org.npsnet.xfsp.datatypes.**XSDBoolean** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDByte** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDComment** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDDouble** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDFloat** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDInteger** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDLong** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDPrimitiveArray** (implements org.npsnet.xfsp.datatypes.SimpleTypeWithName)
- class org.npsnet.xfsp.datatypes.**XSDShort** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDString** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDUnsignedByte** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDUnsignedInt** (implements org.npsnet.xfsp.datatypes.SimpleType)
- class org.npsnet.xfsp.datatypes.**XSDUnsignedShort** (implements org.npsnet.xfsp.datatypes.SimpleType)

- interface org.npsnet.xfsp.datatypes.**SimpleType**
  - interface org.npsnet.xfsp.datatypes.**SimpleTypeWithName**

Interface Hierarchy:

- interface org.npsnet.xfsp.datatypes.**Type**
  - interface org.npsnet.xfsp.datatypes.**SimpleType**
    - interface org.npsnet.xfsp.datatypes.**SimpleTypeWithName**

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[Abilene 00] Dunn,L., Hundertmark,G., Schopis, P., Teitelbaum, B., Turzuanski, M., Wood,R., "Abilene Premium Service Test Program", April 2000.

[Ames 97] Ames, A.L., Nadeu, D.R., and Moreland, J.L., "VRML2.0 Sourcebook", John Wiley& Sons, Inc, 1997.

[BinXML] "Bin-XML™ in for Encoding XML Documents", Technical White Paper, http://expway.fr/graph/Bin-XMLTechnical%20White%20Paper-jan03.pdf March 2003

[Brutzman 97] Brutzman, Don, "Graphics Internetworking : Bottlenecks and Breakthroughs", Addison-Wesley, Reading Massachusetts, 1997, pp. 61-97.

[Brutzman X3D] Brutzman, Don; X3D-Edit Authoring Tool for Extensible 3D (X3D) Graphics, www.web3d.org February 2003

[Capps 00] Capps, M., McGregor, D., Brutzman, D., Zyda, M., "Projects in VR, NPSNET-V, A New Beginning for Dynamically Extensible Networked Virtual Environments", IEEE, October 2000.

[Canterbury 95] Canterbury, Michael; "An Automated Approach to Distributed Interactive Simulation (DIS) Protocol Entity Development", Master Thesis, Naval Postgraduate School, 1995

[CFSM] Lundy, M. Gilbert, "Systems of Communicating Machines: A Model for Communication Protocols", Ph.D dissertation, School of Information and Computer Science, Georgia Institute of Technologies, 1988.

[Cooke 92] Cooke, J.C., Zyda J.M., Pratt, D.R., McGhee, R.B., "NPSNET : Flight Simulation Dynamic Modeling using Quaternions", 1992.

[Cortona 03] Parallel Graphics, http://www.parallelgraphics.com/ February 2003

[DIS] IEEE Standard for Distributed Interactive Simulation – Application Protocols, IEEE Std 1278.1 -1995, (Revision of IEEE Std 1278-1993)

[DOM4JCook 01] Rademacher T, Strachan J, "dom4j cookbook", September 2001 http://www.dom4j.org/cookbook/cookbook.html , December 2003

[DOM4JOnline 03] "dom4j the Flexible Framework for Java" http://www.dom4j.org/ December 2003

[DREN 03] Defense Research & Engineering Network, http://www.hpcmo.hpc.mil/Htdocs/DREN December 2003

[ESS] ESS Model, "The Fastest and Easiest to Use UML Reversing Tool on the Market", http://www.essmodel.com , February 2003

[Fischer 01] Fischer, D. William, "Enhancing Network Communication in NPSNET-V Virtual Environments Using XML-Described Dynamic Behavior Protocols (DBPs), Master Thesis, Naval Postgraduate School, September 2002.

[Ghonaimy 99] Ghonaimy, R. Adeeb, "New Generation Internet and the Evolution Towards Active and Programmable Networks", 16th National Radio Science Conference, 1999.

[Harold 00] Harold, R. Elliotte, "Java Network Programming 2nd Edition", O'Reilly Publications, August 2000

[HLA 00] Kuhl F, Weatherly R, Dahmann J, "Creating Computer Simulation Systems, An Introduction to the High Level Architecture", Prentice Hall PTR, 2000

[HLADMSO] U.S. DoD Defense Modeling and Simulation Office, "High Level Architecture", https://www.dmso.mil/public/transition/hla/ December 2003

[HLAIEEE] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Framework and Rules, IEEE 1516-2000

[Hunter 01] Hunter D., Cagle K., Dix C., "Beginning XML 2nd Edition", Wrox Publishing, February 2002.

[JXTA] Brookshier D., Govoni D., Krishnan N., "JXTA : Java P2P Programming", SAMS Publishing, March 2002.

[Kapolka 02] Kapolka, A., McGregor, D., Capps, M., "A Unified Component Framework for Dynamically Extensible Virtual Environments", ACM, 2002

[Kurose 01] Kurose, F.J., Ross W.K., "Computer Networking A Top-Down Approach Featuring the Internet", Addison Wesley Publications, 2001

[Macedonia 97] Macedonia, R. Michael, Zyda, J. Michael, " A Taxonomy for Networked Virtual Environments", IEEE, 1997

[MBoneInternet2] Almeroth C. Kevin, "The Evolution of Multicast : From the MBone to Interdomain Multicast to Internet2 Deployment", 2000

[McGregor 01] McGregor, D. Kapolka, A. "NPSNET-V : An Architecture for Creating Scalable Dynamically Extensible Networked Virtual Environments". Unpublished presentation given at the 2001 MOVES Institute Open House, Naval Postgraduate School.

[Millau] Girardot, M., Sundaresan,N. " Millau: An Encoding Format for Efficient Representation and Exchange of XML over the Web", http://www9.org/w9cdrom/154/154.html March 2003

[Oasis 02] WAP Wireless Markup Language Specification (WML), http://www.oasis-open.org/cover/wap-wml.html March 2003

[PCWorld 01] Medford, Cassimir, "Pipe Dreams", February 2001.

[Pullen95] Pullen, J. and V. Laviano, "A Selectively Reliable Transport Protocol for Distributed Interactive Simulation," *Proceedings of the 13th Workshop on Standards for the Interoperability of Distributed Simulations*, 1995

[Pullen00] Pullen, J. "Reliable Multicast Network Transport for Distributed Virtual Simulation".

[Salles 02] Salles, J. Ernesto, "The Impact on Quality of Service when Using Security-Enabling Filters to Provide for the Security of Run-time Virtual Environments", Master Thesis, Naval Postgraduate School, September 2002.

[SOAP] Snell J., Tidwell D., Kulchenko P., "Programming Web Services with SOAP", O'Reilly Publications, January 2002.

[Stallings 00] Stallings, William, "Data & Computer Communications 6th Edition", Prentice Hall Publications, June 2000.

[WAP 99] WAP Binary XML Content Format, Version 4-November 1999, http://www1.wapforum.org/tech/documents/SPEC-WBXML-19991104.pdf March, 2003

[WAPW3C] W3C Note (24 June 1999) on "WAP Binary XML Content Format", http://www.w3.org/TR/wbxml/ March, 2003

[W3CSchema] W3C XML Schema, http://www.w3.org/XML/Schema December 2003

[W3CSchema Part-II] W3C XML Schema Part-II, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/ March 2003

[W3CXML] XML Core Working Group Public Page, http://www.w3.org/XML/Core/ December 2003

[Web3D] Web3D Consortium, http://www.web3d.org February, 2003

[Wooldridge01] Wooldridge, Michael; "An Introduction to MultiAgent Systems", Wiley Publications, 2001

[XMLPPM] Compressing XML with Multiplexed Hiearchical PPM Models http://www.cs.cornell.edu/People/jcheney/xmlppm/xmlppm.html March, 2003

[XMLPPM 03] XMLPPM: XML Conscious PPM Compression, http://www.cs.cornell.edu/People/jcheney/xmlppm/xmlppm.html March, 2003

[XMill 00] Liefke,H., Suciu,D., "XMill: an Efficient Compressor for XML Data", In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, paged 153-164, 2000

[XMill 03] An Efficient Compressor for XML, http://www.research.att.com/sw/tools/xmill/ March 2003

[Zeswitz 93] Zeswitz, Steven R., “NPSNET Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Interchange”, Master Thesis, Naval Postgraduate School

[ZLib] “zlib : A Massively Spiffy Yet Delicately Unobtrusive Compression Library”, http://www.gzip.org/zlib/zlib.html March 2003

[Zyda 99] Zyda, M., Singhal S., “Networked Virtual Environments, Design and Implementation”, 1999

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Deniz Kuvvetleri Komutanligi
   Personel Egitim Daire Baskanligi
   Bakanliklar, Ankara
   TURKEY

4. Deniz Harp Okulu Komutanligi Kutuphanesi
   Tuzla, Istanbul
   TURKEY

5. Don Brutzman
   Naval Postgraduate School
   Monterey, California

6. Joseph Sullivan
   Naval Postgraduate School
   Monterey, California

7. Curt Blais
   Naval Postgraduate School
   Monterey, California

8. Mark Pullen
   George Mason University
   Fairfax, Virginia